VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

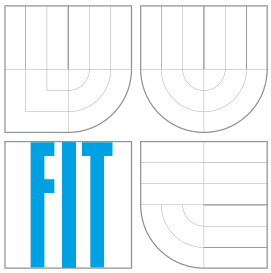
ZOBRAZENÍ SCÉNY POMOCÍ HLUBOKÝCH STÍNOVÝCH MAP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JAN HYPSKÝ
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENÍ SCÉNY POMOCÍ HLUBOKÝCH STÍNOVÝCH MAP

RENDERING USING DEEP SHADOW MAPS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN HYPSKÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. LUKÁŠ POLOK

BRNO 2015

Abstrakt

V počítačové grafice je vykreslování stínů problematické. Aplikace vyžadují volbu vhodného osvětlení a metody stínování s ohledem na rychlost a kvalitu zobrazení. Tato práce je zaměřena na vykreslování stínů a odstranění aliasu, vznikajícího při jejich tvorbě. Pro vytváření stínů bude práce implementovat metody Shadow maps a její rozšířenou variantu Deep shadow maps. Shadow maps umožňují vytvářet stíny nezávisle na složitosti scény a bez ošetření aliasu. Deep shadow maps dovedou zobrazit stíny spolu s odstraněním aliasu, čímž se vylepší kvalita stínování. Pro osvětlení scény je použit Lambertův lokální osvětlovací model. Výsledná aplikace je vytvořena s využitím knihovny OpenGL.

Abstract

In computer graphics is problematic shadow rendering. Applications require to choose the appropriate lighting and shading method with respect to speed and display quality. This work is focused on rendering the shadows and eliminate alias, arising during their creation. For to create shadows will work to implement the methods Shadow Maps and she's extensions variant Deep Shadow Maps. Shadow maps allow to create shadow independently of the scene complexity without treatment alias. Deep shadow maps are able to display shadows along with the removal of alias thereby improve the quality of shading. For lighting scene are used local Lambert lighting models. The resulting application is created with using library OpenGL.

Klíčová slova

alias, metody stínování, stínové mapy, hluboké stínové mapy, lokální osvětlovací modely, OpenGL

Keywords

alias, shading method, shadow maps, deep shadow maps, local lighting models, OpenGL

Citace

Jan Hypský: Zobrazení scény pomocí hlubokých
stínových map, bakalářská práce, Brno, FIT VUT v Brně, 2015

Zobrazení scény pomocí hlubokých stínových map

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Lukáše Poloka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Hypský
19. května 2015

Poděkování

Za výborné vedení, odborné připomínky a celkovou podporu při tvorbě této bakalářské práce bych rád poděkoval vedoucímu práce panu Ing. Lukášovi Polokovi.

Obsah

Kapitola 1

Úvod

Zobrazení stínů průhledných a neprůhledných těles je v počítačové grafice velmi komplikované. Aby bylo možné stíny vykreslit, musíme nejdříve zvolit vhodnou metodu osvětlení. Scéna může být osvětlena lokálními zdroji světla, poskytující rovnoměrnou intenzitu světla, nebo globálními zdroji světla.

Lokální modely [14] osvětlení nezahrnují do svých výpočtů osvětlení okolí, mají snadnou implementaci a poskytují velmi rychlé vykreslení scény. Zaměřují se především na rychlost a proto většinou nejsou matematicky ani fyzikálně podloženy, ale vznikají, jako výsledek pokusů a experimentů. Světla v těchto modelech mají konstantní intenzitu a jsou vyzařována bodovými zdroji. To má za následek vznik ostrých stínů. Mezi zástupce těchto modelů patří Lambertův nebo Phongův model. Globální modely jsou oproti lokálním modelům založeny na fyzikálních zákonech. Do celkového osvětlení počítají i světlo odražené z okolních objektů scény (okolní osvětlení). To umožňuje vykreslit scénu velmi realisticky za cenu nižší rychlosti vykreslení. Jednou z technik globálního osvětlení je technika *Light Propagation Volumes* [8].

Pro tvorbu stínu existuje v počítavé grafice velké množství metod. Některé metody poskytují velmi kvalitní výsledky, ale za cenu snížení celkového výkonu. Často bývají i závislé na zvoleném osvětlení nebo na složitosti scény. To má za následek uplatnění jen ve velmi specifických aplikacích. Existují však i techniky tvorby stínů, které nejsou závislé na složitosti scény. Jednou z těchto technik jsou *stínové mapy* (shadow maps) [3].

Stínové mapy (SM) jsou jednoduché a snadno se implementují. Princip stínových map je možné popsat několika kroky. V prvním kroku dochází k vykreslení scény z pozice zdroje světla a vytvoří se hloubková mapa. Následně se do hloubkové mapy zapisují hodnoty vzdálenosti nejbližších viditelných povrchů od zdroje světla. V druhém kroku dochází k vykreslení scény z pozice pozorovatele. U každého fragmentu je následně zjištěna vzdálenost od zdroje světla a porovnána s hodnotou, která je uložena v hloubkové mapě. Pokud se hodnota rovná hodnotě z hloubkové mapy, je fragment osvětlen. V opačném případě není fragment osvětlen a nachází se ve stínu. Použití stínových map přináší i velkou nevýhodu v podobě aliasingu. Klasická stínová mapa navíc nedokáže vykreslit stíny průhledných těles. Pokud je třeba tyto stíny zobrazit, musíme stínovou mapu modifikovat. Jednou z těchto modifikací, umožňujících zobrazit složité stíny, je tzv. *deep shadow map* (hluboká stínová mapa) [9].

Deep shadow maps (DSM) jsou většinou složeny z několika vrstev. Do těchto vrstev se následně ukládají informace o viditelnosti a barvách objektů scény. Možnosti vykreslení různých typů stínů závisí na počtu vrstev DSM. Hluboké stínové mapy mají větší pamětové i výpočetní nároky, než klasické stínové mapy. V porovnání s klasickou stínovou mapou však dosahují velmi kvalitních výsledků v zobrazení stínů (obrázek 2.12). Deep shadow maps

umožňují vykreslit stíny drobných objektů scény (tráva, srst), objemových těles (kouř, mlha) nebo objektů v pohybu. Poradí si také s barevnými stíny průhledných těles. Výsledkem bývá velmi kvalitní a realistické stínování. Hluboké stínové mapy lze popsat jako pole pixelů, ve kterém si každý pixel ukládá hodnotu funkce viditelnosti. Tato viditelnost je definována paprskem světla procházejícím skrze pixel. Počáteční hodnota viditelnosti je nastavena na maximum („1“). Pokud paprsek narazí na těleso blokující jeho průchod je tato hodnota snížena. V případě, že se viditelnosti dostane na hodnotu „0“, je tento papsek zablokován.

U stínových map se objevují i nežadoucí artefakty snižující celkovou kvalitu stínování. Jedním z nejčastějších problémů stínových map je vznik tzv. aliasu. Tento jev vzniká při diskretizaci spojitých prvků, kdy je několik vzorků obrazu promítáno do stejného texelu stínové mapy. Alias se nejčastěji projevuje při vykreslení stínů zkosených hran. Metod pro odstranění nebo snížení aliasu existuje několik. Lze například zvětšit rozlišení stínové mapy, čímž se sníží pravděpodobnost promítaní vzorků obrazu na stejné místo stínové mapy. Pro hluboké stínové mapy je vhodnější použití techniky analytického antialiasu hran. Zjištění všech obrysových hran lze například použitím metody *okřídlených hran* [4]. Následně jsou hrany vykresleny s antialiasem, kdy se tento antialias projeví i na stínech těles.

Kapitola 2 bude zaměřena na teoretický rozbor práce. Nejdříve budou popsány hlavní typy a modely osvětlení scény 2.1. Další podkapitola představí metody stínování shadow maps 2.2.1 a především deep shadow maps 2.2.2. V závěru této části budou popsány hlavní prvky OpenGL (2.3), použité pro tvorbu stínových map v této práci. Seznámení s požadavky na výslednou aplikaci a jejich začlenění do návrhu aplikace je v části 3. Následující kapitola 4 se zaměří na implementaci aplikace. Nejdříve budou představeny funkce pro načítání a zpracování scény. Dále jsou popsány hlavní struktury aplikace. Konec kapitoly přiblíží základní ovládání aplikace spolu s rozebráním problémů vniklých při tvorbě programu. Dosažené výsledky jsou v kapitole 5. Zhodnocení výsledků v rámci tvorby hlubokých stínových map spolu s dalšími možnostmi vylepšení kvality proběhne v kapitole 6.

Kapitola 2

Teoretický rozbor

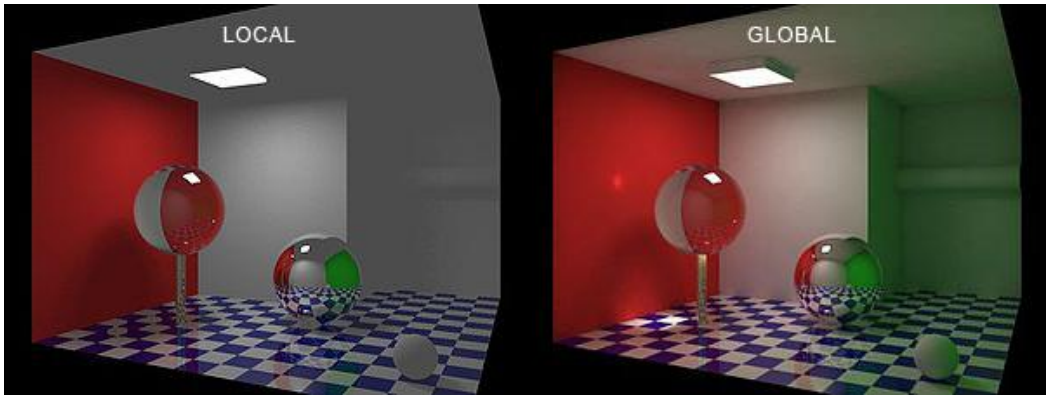
V následující kapitole budou popsány principy lokálního osvětlení spolu s popisem základních modelů lokálního osvětlení. Následně budou vysvětleny metody stínových map (Shadow Maps) a hlubokých stínových map (Deep Shadow Maps), tvořícím základ této práce. Závěr kapitoly je věnován hlavním prvkům OpenGL.

2.1 Osvětlovací modely

V kontextu počítačové grafiky lze osvětlovací model popsat jako intenzitu světla v různých bodech scény. Intenzitu světla lze definovat jako světelný tok dopadající na plochu. Mezi nejdůležitější faktory ovlivňující model patří vlastnosti osvětlovaných objektů, typ světelného zdroje a vzdálenost od pozorovatele. Modely můžeme rozdělit podle vlastností do dvou hlavních skupin: na lokální a globální modely. Některé modely nemusí být založeny na fyzikálních vlastnostech. Především u lokálních modelů hraje větší roli lidské vnímání. Při tvorbě této kapitoly a jejich podkapitol byly využity informace z [5], [11] a [14] pokud není stanoveno jinak.

2.1.1 Porovnání typů osvětlení

Lokální modely neprovádí výpočet okolního osvětlení. Charakteristickými vlastnostmi jsou jednoduchost, velká rychlost vykreslení a především vizualizace soustředěná na vnímání pozorovatele (nemusí se vždy řídit fyzikálními zákony). Uplatnění nalézají u aplikací, kde je kladen důraz na krátkou dobu vykreslení (např. hry). Naproti tomu globální modely jsou vždy založeny na fyzikálních zákonech. Pracují s okolním světlem, proto při vykreslení scény dosahují vynikajících výsledků co se týká kvality zobrazení. Nevýhodou těchto modelů je jejich složitost a časová náročnost na výpočet scény. Rozdíl lokálního a globálního osvětlení je patrný z obrázku 2.1.



Obrázek 2.1: Vizuální rozdíl mezi lokálním a globálním osvětlením
[1]

2.1.2 Lokální modely

Lokální modely nepracují s okolním osvětlením (nepřímým osvětlením z osvětlených objektů scény), ale pouze se vztahem mezi světelnými zdroji a objekty. Hlavní využití těchto modelů je především v aplikacích zaměřených na plynulé zobrazení scény nebo rychlé zobrazení vysokého množství dat. Zdroje světla proto musí být jednoduché. Řešením je použití bodových světel. Bodová světla ve skutečnosti neexistují, ale mnoho zdrojů, lze pomocí nich aproximovat. Tato světla vytvářejí ve scéně ostré stíny. Intenzitu osvětlení můžeme rozdělit na dvě základní složky:

- **Difúzní (Diffuse)** - Složka světla, která se po dopadu na povrch objektu rovnoměrně rozptýlí do všech stran. Pokud budeme viditelný bod pozorovat z jakéhokoliv úhlu, difúzní složka bude vždy stejná. Intenzita je závislá pouze na úhlu dopadajícího světla.
- **Odrazová (Specural)** - Odrazová složka je závislá na bodu pozorovatele a vlastnostech materiálu povrchu. Tyto vlastnosti lze popsat například BRDF funkcemi.

Jelikož v lokálních modelech neuvažujeme osvětlení odražené od okolních objektů, výsledná scéna je velmi tmavá. U většiny modelů se problém tmavé scény vyřešil přidáním další složky intenzity osvětlení nazvané okolní (ambient). Výsledná rovnice pro obecný popis lokálních modelů (2.1) je složena ze všech tří intenzit světla. Difúzní a odrazovou složku lze spočítat a vyhodnotit pro každý světelný zdroj odděleně. Ambientní složka je většinou zastoupena konstantou zvyšující jas scény.

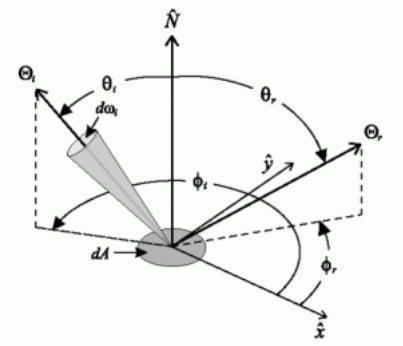
$$L(x, e) = I_d(x) + I_s(x, e) + I_a, \tag{2.1}$$

kde $L(x, e)$ je výsledná intenzita osvětlení daného bodu x, pozorovaného z bodu e,$I_d(x)$ je difúzní složka osvětlení v bode x, $I_s(x, e)$ je odrazová složka osvětlení v bodě x, pozorovaného z bodu e a I_a je složka simulující světlo odražené z okolí.

[1] Převzato dne 14.1.2015 z: http://www.thegnomonworkshop.com/news/2013/05/light-for-3d-animation-globally-illuminating-the-job-of-lighting-artists/

BRDF funkce

Bidirectional reflectance distribution function (Dvousměrná distribuční funkce) popisuje fyzikální i matematické vlastnosti odrazu na povrchu osvětleného tělesa (obrázek 2.2). Formálně lze BRDF definovat, jako poměr rozdílu záření odraženého v odchozím směru a rozdílu záření z ostatních směrů na povrch bodu x [14].



Obrázek 2.2: Zobrazení úhlu dopadajícího a odraženého paprsku od povrchu tělesa.
[2]

Funkce závisí na pozici pozorovatele, poloze všech zdrojů světla a na vlnové délce světla. Paprsky různých vlnových délek mají na povrchu objektu rozdílné chování (odlišně se odrážeží i pohlcují). Rovnice 2.2 zobrazuje obecné vyjádření BRDF [14].

$$\rho(\Theta_i, \Theta_r, \lambda) = \frac{dL_r(\Theta_i, \Theta_r, \lambda)}{dE_i(\Theta_i, \lambda)}, \tag{2.2}$$

kde Θ_i je elevace a azimut (směr šíření) dopadajícího paprsku, Θ_r je elevace a azimut (směr šíření) odraženého paprsku, λ reprezentuje vlnovou délku světla,L je záření a E je ozáření plochy.

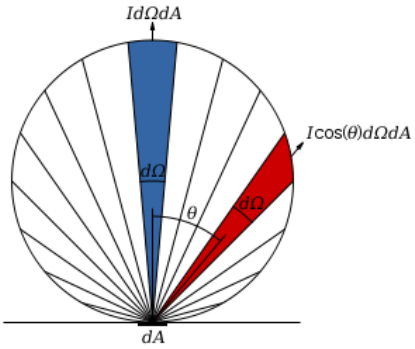
Lambertův model

Lambertův model odrazu patří mezi základní modely pro zobrazení dokonalého difúzního odrazu. Popisuje takzvané Lambertian surface (Lambertovy povrchy), které rovnoměrné rozptylují světlo do všech směrů a současně se řídí *Lambertovým kosinovým zákonem.*

Lambertův kosinový zákon [2] (známý též, jako kosinový emisní zákon) říká, že intenzita záření nebo svítivosti, pozorovaná z odrazu ideálního difúzního povrchu nebo tělesa, je přímo úměrná kosinu úhlu θ mezi směrem dopadajícího světla a kolmicí povrchu (obrázek 2.3).

Obvykle se při vykreslování v reálném čase ρ_d/π nahrazuje difúzní barvou (C_d). Díky této substituci můžeme následně z rovnice vypustit normalizační faktor $1/\pi$. Potom povrch, osvětlený dopadem paprsků ($cos\theta_i = \langle N, L\rangle = 1$) od světelného zdroje s intezitou světla $I = 1$, zobrazuje přesně barvu C_d. Dostáváme rovnici pro výpočet Lambertova modelu (2.3). Tato skutečnost se využívá v grafických aplikacích při specifikaci barvy vrcholu nebo vykreslování do textur.

[2] Převzato dne 13.1.2015 z: http://math.nist.gov/~FHunt/appearance/brdf.html



Obrázek 2.3: Lambertův kosinový zákon.
[3]

$$I_o = I_i . \langle N, L \rangle . C_d, \tag{2.3}$$

kde I_o je intenzita odraženého světla, I_i je intenzita příchozího světla, $\langle N, L \rangle$ reprezentuje kosinový faktor normálového vektoru a normalizovaného světelného vektoru a C_d je difúzní barva.

Lambertův model se v praxi používá, jako difúzní komponenta jiných stínových modelů, zaměřených na modelování odrazu. Díky své efektivitě výpočtu, snadnému nastavení parametrů (nastavuje se pouze difúzní barva) a zachycení vzhledu mnoha difúzních materiálů nalézá využití v grafických aplikacích. Samostatně se tento model prakticky nepoužívá.

Phongův model

Za nejpoužívanější lokální model osvětlení v počítačové grafice je označován Phongův model. Předností tohoto modelu je nenáročnost, poměrně snadná implementace, jednoduché použití a zobrazení lesklých zrcadlících se ploch, které přináší kvalitní výsledky vykreslení. Tento model představil Bui Tuong Phong ve své disertační práci [10]. Model využívá kosinu úhlu mezi zářením odraženém v odchozím směru (ω_o) a směrem dokonalého lesklého odrazu zvýšeném na úroveň energie e [4]. Výsledná BRDF (2.4) funkce pro Phongův model se zapisuje jako:

$$f_\lambda(x, \omega_o, \omega_i) = k_s C_s \frac{\cos^e d}{\cos \theta_i} + k_d C_d \tag{2.4}$$

Z rovnice (2.4) je patrné, že Phongův klasický model ruší kosinus faktoru místního integrálu odrazu. Při zvyšování exponentu e se zúží tzv. spekulární lalok a naopak při snížení exponentu se spekulární lalok rozšiřuje. Původně model nastavoval barvu spekulárního světla s důrazem na barvu dopadajícího světla. Následkem bylo vytvoření syntetického, plastického vzhledu, který byl velmi často kritizován. Dnes se už model obvykle používá s oddělením lesklé barvy [11]. Jelikož jsou splněny požadavky BRDF, je takto definovaný model fyzikálně korektní (obrázek 2.4).

[3] Převzato dne 13.1.2015 z: http://en.wikipedia.org/wiki/Lambert%27s_cosine_law

Obrázek 2.4: Schéma Phongova modelu. a) BRDF vyjádření, b) empirické vyjádření. [4]

Zdroj světla je zastoupen symbolem slunce a pohled pozorovatele symbolem kamery. Jednotlivé vektory mají tento význam: L světlo dopadajícího na objekt, V směr pohledu uživatele, N vektor normály povrchu a R spekulární odraz. Phongův model je možné definovat empiricky (obrázek 2.4). Výsledné odražené osvětlení modelu (2.5) je složeno z ambientního, difúzního a spekulárního osvětlení [5].

$$I = I_a + I_d + I_s, \tag{2.5}$$

kde I je výsledné osvětlení, I_a je ambientní odraz vypočítaný, jako součin intenzity okolního osvětlení (I_A) a koeficientu odrazivosti materiálu (k_A)$\Rightarrow I_a = I_A k_A$, I_d reprezentuje difúzní osvětlení vypočítané, jako součin intenzity difúzního osvětlení (I_D) a koeficientu odrazivosti materiálu (k_D) $\Rightarrow I_d = I_D k_D$ a I_a zastupující spekulární osvětlení vypočítané jako součin barevných složek dopadajícího paprsku (I_L), koeficientu zrcadlové odrazivosti materiálu (k_S) a skalárního součinu spekulárního odrazu (R) a směru pohledu uživatele (V), kdy je skalární součin umocněn mírou lesklosti (n) $\Rightarrow I_s = I_L k_S (R \cdot v)^n$. Rovnice (2.5) platí pouze pro empirické vyjádření Phongova modelu, který není zcela fyzikálně správný. Nedodržuje například zákon o zachování energie a požadavky na vlastnosti BRDF [11].

2.1.3 Globální modely

Pro zobrazení velmi realistického vzhledu scény je zapotřebí počítat s okolním osvětlením, které se tvoří odrazem od povrchů objektů. Lokální modely na tento typ zobrazení nejsou vhodné. Jsou příliš jednoduché, nebývají založené na matematicko-fyzikálních zákonech a ve svých výpočtech s okolním světlem vůbec nepočítají nebo ho nahrazují konstantní hodnotou. Proto je zde lepší používat globální modely, které bývají složitější na implementaci, ale zahrnují do výpočtu osvětlení scény i okolní světlo [14] [5]. Nejznámějšími představiteli jsou metody *Ray Tracing*, *Radiosity* 2.1.3 a *Light Propagation Volumes* 2.1.3.

Radiozita

Poprvé byla tato metoda představena v roce 1984 [6]. Radiozita je zde definována, jako polokulový integrál energie opouštějící povrch. Z pohledu pozorovatele se zdá, že povrch (j) vyzařuje tok energie (B_j) z imaginárního povrchu, složeného ze dvou hlavních částí 2.6.

$$B_j = E_j + p_j H, \tag{2.6}$$

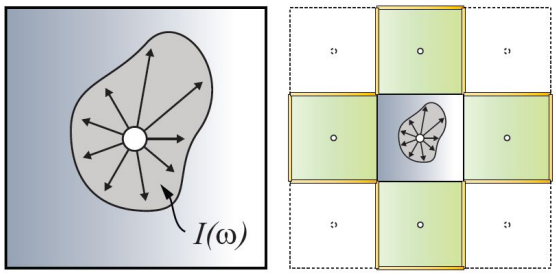
[4] Převzato dne 13.1.2015 z: http://cs.wikipedia.org/wiki/Phong%C5%AFv_osv%C4%9Btlovac%C3%AD_model

kde B_j je radiozita povrchu j (celková míra energie záření , která opustí povrch za jednotku času a plochy $[W/m^2]$, E_j míra emisí přímé energie z povrchu j za jednotku času a plochy $[W/m^2]$, p_j je reflektivita povrchu j a přestavuje zlomek světla odraženého zpět do polokulovitého prostoru, H_j zastupuje energii záření přicházející na povrch objektu za jednotku času a plochy $[W/m^2]$ [6].

Z pohledu pozorovatele není možné rozlišit složky na pravé straně rovnice (2.6), protože obě mají v místě stejnou směrovou distribuci (difuzní). Není proto nutné od sebe oddělovat odražené a vyzařované světlo. Oproti většině lokálních modelů není vzorec výpočtu radiozity empirický. Radiozita se řídí fyzikálními zákony (především zákonem o zachování energie) a lze ji popsat pomocí distribuční funkcí BRDF [6].

Light Propagation Volumes

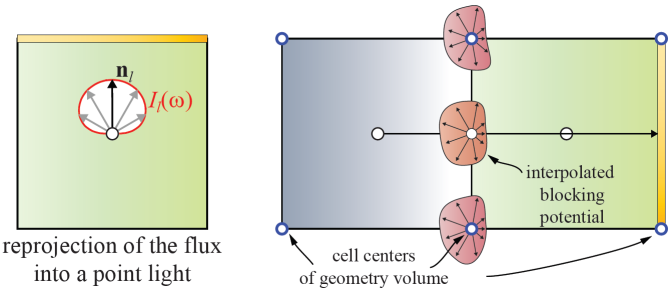
Tato metoda provádí výpočet nepřímého osvětlení ve scéně. Light propagation volumes [8] (LPV) vychází z diskrétních souřadnicových metod a *Lattice-Boltzmanovy* techniky osvětlení, kdy je světlo ve scéně reprezentováno mřížkou vzorků. Pochody světla lze pak modelovat pomocí jednoduchých operací, které lze i paralelizovat. V každé buňce matice je uložena intenzita světla. Použití mřížek vzorků s sebou přináší zásadní nedostatky. Vzniká takzvaný ray effect, který způsobuje lehké rozmazání světla. Tento problém se řeší využitím dvou mřížek. V jedné mřížce je uložena intenzita inicializovaná na povrchu osvětleném nepřímým nebo nízkofrekvenčním přímým osvětlením. Druhá mřížka ukládá přibližný objem světla, který prochází skrz scénu. Celkový výpočet LPV lze rozdělit do několika kroků.



Obrázek 2.5: a) Buňka se zdrojem směrové intentizy uprostřed. b) Šíření světla do okolních buněk v axionálním směru [8].

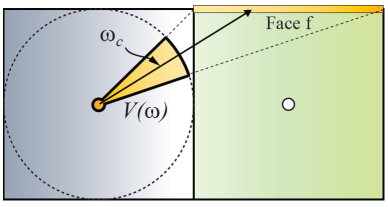
Nejříve se provádí inicializace LPV mřížky pro povrchy, které vytváří nepřímé osvětlení. Vytvoří se virtuální bodové zdroje světla (VPLs) reprezentující nepřímé osvětlení pomocí reflexivní stínové mapy (RSM). Jedná se o vylepšenou klasickou stínovou mapu, která zaznamenává povrchy osvětlené přímým světlem, vytvářející první odraz světla. Každý texel p reflexivní mapy lze intepretovat jako oblast zdroje světla. Zdroj má svou spektrální a směrovou intenzitu $I_p(\omega)$ (obrázek 2.5). Tuto intenzitu můžeme vypočítat, jako $I_p(\omega) = \Phi_p < n_p|\omega >_+$, kde $< .|. >$ zastupuje skalární součin, n_p je normála textelu p, Φ_p reprezentuje světelný tok p a ω určuje požadovaný směr šíření. Virtuální zdroje světla jsou následně tranformovány do sférické harmonické reprezentace (SH). Pozice zdrojů v mřízce jsou zjištěny z mapy. Směřuje-li normála virtuálního zdroje světla mimo buňku(od jejího

středu), potom přínos světla toho zdroje patří vedlejší buňce. Pokud by se tento přínos přidal k současné buňce, mohlo by docházet k vlastnímu zástínění nebo osvětlení buňky. Aby se těmto jevům zabránilo, VPL se posune o polovinu velikosti buňky ve směru své normály. Tento posuv se musí uskutečnit ještě před samotným určením pozice v mřížce. Výsledná hodnota směrové distribuce intenzity každé buňky je výjadřena SH n^2 koeficienty c_l, m, získanými z n SH polí a základními funkcemi $y_l, m(\omega)$. Obecně pak platí $-l \leq m \leq l$, kde l zastupuje index pásma a m úroveň pásma. V případě, že buňka je ovlivněna více VPL jsou hodnoty těchto koeficientů sečteny. Výpočet těchto koeficientů se provádí pro celé spektrální pásmo [8].



Obrázek 2.6: Objemové zastoupení těles scény s vyznačeným blokování šíření světla [8].

V dalším kroku je potřeba vytvořit objemové zastoupení povrchů scény (GV). Pro toto zastoupení se vytváří nová mřížka, která zachycuje blokování světla díky tělesům ve scéně a také umožňuje výpočet nepřímých stínů. Velikost GV mřížky je totožná jako LPV. Podobná je i inicializace mřížky, kdy jsou vzorky získávány při tvorbě pohledu kamery a RSM pro velké povrchy scény. Každý ze vzorků přestavuje element povrchu (tzv. surfel). Surfel nese informace o umístění, orientaci a velikosti. Průchod světla je tímto surfelem blokován. Velikost blokování závisí na velkosti surfelu a úhlu, který svírá světlo s normálou surfelu. Pravděpodobnost zablokování světla jedním surfelem za předpokladu, že všechny objekty scény jsou uzavřené plochy, je $B(\omega) = A_s * s^-2 < n_s|\omega >_+$, kde A_s vyznačuje oblast surfelu, s zastupuje velikost buňky, n_s je normála surfelu a ω směr šíření světla. Rozdílné oproti LPV je však uložení buňek. GV mřížka má buňky posunuté o poloviční velikost buňky tak, že středy těchto buněk se nacházejí v rozích LPV mřížky (obrázek 2.6. To přináší při šíření světla možnost lepší interpolace potenciálu blokování. Výsledná mřížka dokáže vytvářet měkké stíny. Pokud však mřížka pracuje s velmi malými porvchy (např. tráva) nezabírajícími ani jednu celou buňku mřížky, stíny se nevytvoří [8].



Obrázek 2.7: Zobrazení světelného toku dopadajícího na stěnu cílové buňky [8].

Intenzita světelného šíření procházejícího scénou se vypočítá pomocí lokálních iteračních kroků. Pro první iteraci je na vstup přivedena LVP mřížka z fáze inicializace. U všech

ostatních iterací se na vstup přivádí mřížka LPV z předchozí iterace. Buňky ukládají intenzitu světla jako sférický harmonický vektor (SH vector) Přibližná hodnota této intenzity je $I(\omega) = \sum_l, mc_l, my_l, m(\omega)$. Světelné šíření pak probíhá ve směru hlavních os 2.5. Pro výpočet světelného toku dopadajícího na stěny f přilehlých buněk zavádíme funkci viditelnosti $V(\omega)$. Tato funkce nabývá hodnoty $V(\omega) = 1$ v případě, že paprsek ze zdroje ve středu zdrojové buňky protíná stěnu f přilehlé buňky a hodnoty $V(\omega) = 0$ pokud stěnu f neprotíná. Ukázka funkce viditelnosti pro horní stěnu přilehlé buňky je na obrázku 2.7. Pomocí viditelnosti lze pak světelný tok, dopadající na stěnu buňky, vyjádřit jako $\Phi_f = \int_\Omega I(\omega)V(\omega)d_\omega$. Integrál intenzity světla a funkce viditelnosti lze vypočítat pomocí skálarního součinu vectorů c_l, m a y_l, m. Pro vysoké řády SH je vypočtený integrál dostatečně přesný, nicméně u nižších řádů může docházet k nepřesnostem. Proto pro každý face f přilehlé buňky vypočítává prostorový úhel $\Delta\omega_f = \int_\Omega V(\omega)d\omega$ a vybere se centrální směr ω_c. Výslední zářivý tok Φ_f se pak vypočítá $\Phi_f = \delta\omega_f/(4\pi).I(\omega_c)$. Velikost celkové intenzity je pak ovlivněna pravděpodobností blokování překažkou (získanou z mřížky GV bi-lineární interpolací). Dále je potřeba vytvořit zdroj světla ve středu přilehlé buňky, který mám stejný světelný tok Φ_f jako tok přijatý z bodového světla šířením Φ_l a směřuje ke stěně f. Podobně jako při inicializaci světla se u zdroje hromadí SH koeficienty. Šíření světla z tohoto zdroje je pak vypočítáno pro každou stěnu sousedních buněk. Výsledky všech iterací udávají celkové světlo nacházející se ve scéně. Nyní stačí získávat intenzitu trilineární interpolací SH koeficientů a tuto intenzitu převést na záření [8].

Tato metoda není v této práci implementovaná. Je zde uvedena především, jako možné budoucí rozšíření stávající práce, kdy by nahradila současnou metodu osvětlení pro své kvalitní výsledky 2.8.

Obrázek 2.8: Osvětlení scény pomocí LPV [8].

2.2 Stínování

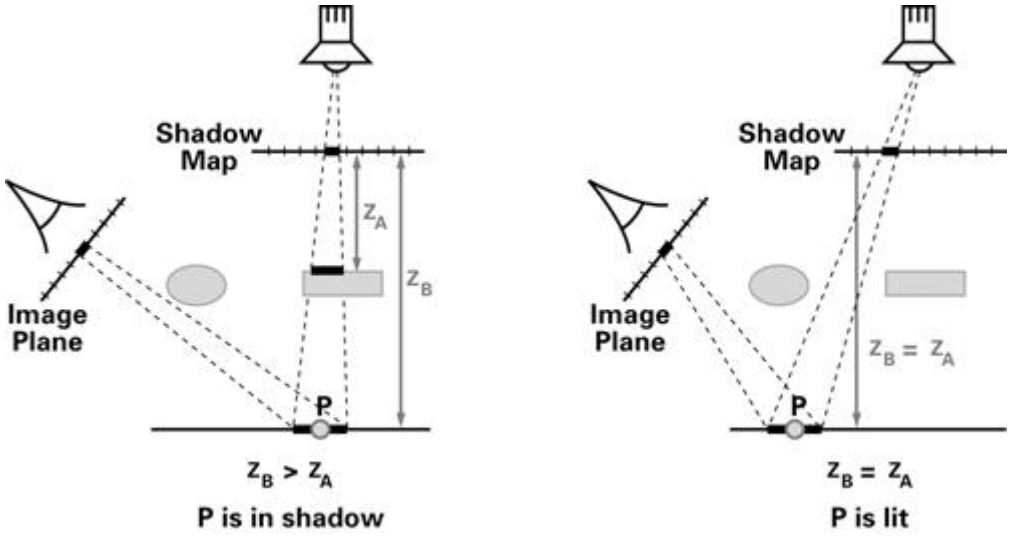
Metod, zabývajících se tvorbou stínu v počítačové grafice, je velké množství. Tato kapitola se bude zabývat pouze dvěma z nich - *Shadow maps* 2.2.1 a *Deep shadow maps* 2.2.2.

2.2.1 Shadow maps

První zmínka o stínových mapách se objevila v článku Lance Wiliamse [13]. Tradiční stínové mapy jsou založeny na algoritmu ukládání hodnot hloubky viditelného povrchu (Z-buffer)

a vytvoření mapy hloubky (depth map), potřebné pro výpočet stínu. Hlavní výhodou stínových map je jejich nezávislost na složitosti scény. Jejich algoritmus je jednoduchý a velmi podobný algoritmu viditelnosti objektů, implementovaném v grafických adaptérech [3].

Algoritmus se skládá z několika kroků. Nejdříve se vykreslí scéna z pozice zdroje světla. Vznikne hloubková mapa (depth map), označovaná též jako *stínová mapa*. Tato mapa ukládá hodnoty vzdálenosti nejbližšího pixelu od zdroje světla (hodnota hloubky Z). Následně je scéna vykreslena z pozice pozorovatele. U každého fragmentu je zjištěna hodnota vzdálenosti od zdroje světla a porovnána s hodnotou v hloubkové mapě. Pokud je vzdálenost fragmentu od světla větší, než hodnota uložená v mapě hloubky je fragment ve stínu. Je-li vzdálenost rovna údaji v mapě hloubky je fragment osvětlen [3]. Postup algoritmu je ilustrován obrázkem 2.9.



Obrázek 2.9: Algoritmus výpočtu viditelnosti objektu ve stínových mapách. Vlevo - Bod P se nachází ve stínu jiného tělesa. Vpravo - Bod P je přímo osvětlen zdrojem světla. [6]

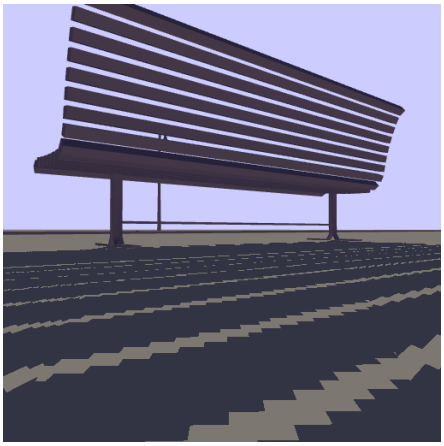
Alias

Stínové mapy pracují v prostoru obrazu a proto je jejich rozlišení omezené. Některé vzorky obrazu jsou proto promítány do stejného texelu stínové mapy. To přináší velikou nevýhodu v podobě vzniku nežádoucích artefaktů tzv. aliasing. Aliasing vzniká diskretizací spojitých prvků. Typicky se tento jev projeví u zkosených hran vytvořením *stair-stepping* artefaktů (obrázek 2.10). U většiny aplikací se tento problém řeší zvětšením rozlišení stínové mapy. Pravděpodobnost vzniku aliasingu se tím sníží, ale není odstraněna. Navíc se tímto krokem zvýší požadavky na paměť a výpočetní výkon [3] [7].

2.2.2 Deep shadow maps

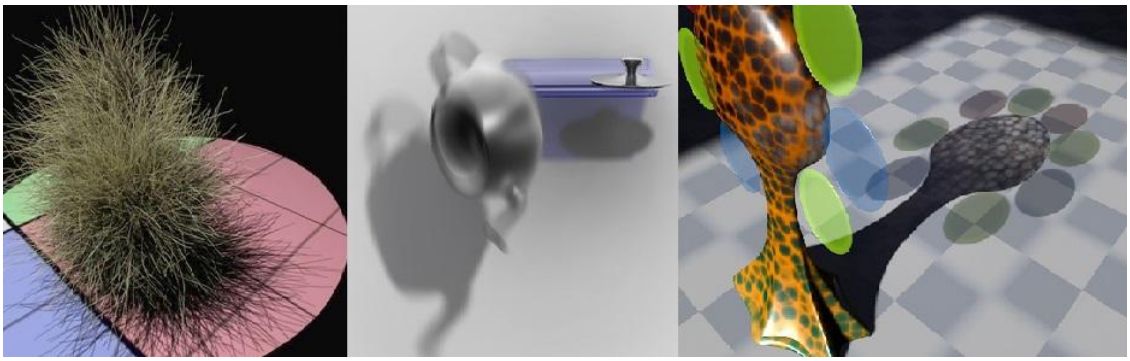
Deep shadow maps (DSM) [9] můžeme popsat jako pole pixelů, ve kterém si každý pixel pole ukládá funkci viditelnosti. Oproti tradičním shadow maps mají deep shadow maps několik

[6] Převzato dne 15.1.2015 z: http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter09.html



Obrázek 2.10: Ukázka aliasingu vznikajícího u stínových map.
[7]

výhod. Hluboké stínové mapy jsou časově i prostorově více složité než klasické stínové mapy, ale zobrazují stíny drobných i objemových těles (např. srst, vlasy, kouř, mlha), poradí si se stíny těles v pohybu (*motion blur*) , zobrazují i barevné stíny a umožňují zobrazení stínů poloprůhledných objektů (2.11). Tyto výhody poskytují zobrazení velmi kvalitních a realistických stínů, které klasické stínové mapy nedosahují (2.12).

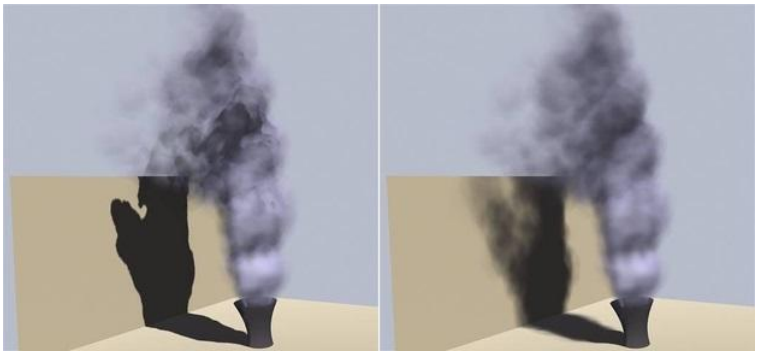


Obrázek 2.11: Zobrazení stínů pomocí deep shadow maps: 1. chomáč vlasů, 2. těleso v pohybu, 3. průhledné barevné těleso.
[8]

Definice a princip DSM

Funkce viditelnosti těchto map je definována světelným paprskem, který prochází pixelem a začíná na stínové kameře. Potom se hodnota funkce při dané hloubce vypočítá jako zlomek původního výkonu světelného paprsku, pronikajícího danou hloubkou. V každém pixelu má funkce viditelnosti počáteční hodnotu nastavenu na „1“ a reprezentuje propouštěné světlo.

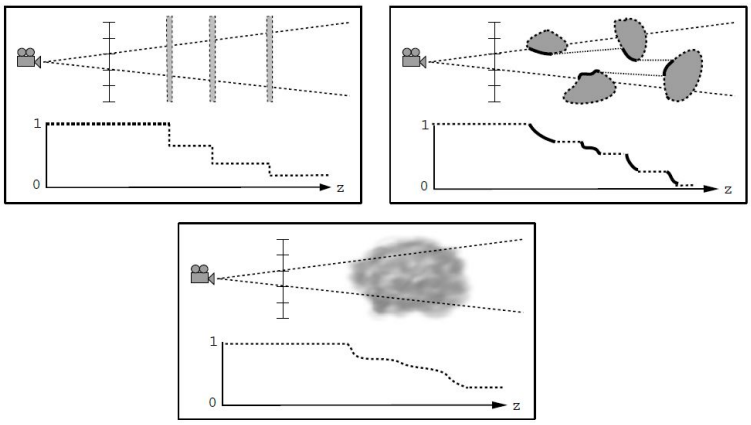
[7] Převzato dne 15.1.2015 z: http://gamma.cs.unc.edu/LOGSM/
[8] Převzato dne 15.1.2015 z: http://www.hradec.com/ebooks/CGI/RMS_1.0/mtor/rendering/feature-guide/DeepShadows.html



Obrázek 2.12: Rozdíl v kvalitě stínů - klasická shadow map (vlevo), hluboká stínová mapa(vpravo).

[9]

Pokud je průchod světla blokován, hodnota se snižuje („0“ - veškeré světlo je zablokováno) (2.13) [9].



Obrázek 2.13: Zobrazení funkce viditelnosti a změny energie paprsku. (Vlevo) Energie paprsku snížená průchodem poloprůhlednými tělesy. (Vpravo) Neprůhledná tělesa. (Dole) Objemový útlum vlivem kouře [9].

Definici můžeme zpřesnit, pokud budeme uvažovat paprsek v počátku stínové kamery a procházející bodem na rovině obrazu. Světlo paprsku pak bude tlumeno plochou nebo objemem těles a absorbováno. Podíl pronikajícího světla do hloubky, známe jako funkci propustnosti. Pro každý pixel se funkce viditelnosti získá filtrací nedaleké funkce propust-

[9]Převzato dne 15.1.2015 z: http://www.hradec.com/ebooks/CGI/RMS_1.0/mtor/rendering/feature-guide/DeepShadows.html

nosti a následným převzorkováním na pixel ve středu (2.7) [9].

$$V_i, j(z) = \int_{-r}^{r} \int_{-r}^{r} f(s,t)\tau(i + \frac{1}{2} - s, j + \frac{1}{2} - t, z)ds\ dt, \tag{2.7}$$

kde V_i, j představuje funkci viditelnosti, τ je funkce propustnosti, zastupuje bandlimiting filtru pixelu, r je poloměr filtru a $(i + \frac{1}{2}, j + \frac{1}{2})$ je pixel ve středu.

Oproti klasické filtraci obrazu se tato filtrace aplikuje na každou hodnotu odděleně. Funkce viditelnosti jsou úzce spojeny s *alfa kanály*. Alfa kanál reprezentuje tlumení světla v důsledku dvou poloprůhledných povrchů a částečného pokrytí pixelů. Kanál ukládá pouze jednu hodnotu na pixel, která odpovídá světlu blokovanému v rovině. Vztah mezi alfa kanály a funkcí viditelnosti je zobrazen rovnici (2.8). U Deep shadow map je výpočet přibližné hodnoty roven ve všech hloubkách a výsledek je uložen jako funkce $1 - \alpha$. Díky tomu obsahuje pixel informace o útlumu a pokrytí pro každou hloubku stínové mapy [9].

$$\alpha_{i,j} = 1 - V_{i,j}(\infty) \tag{2.8}$$

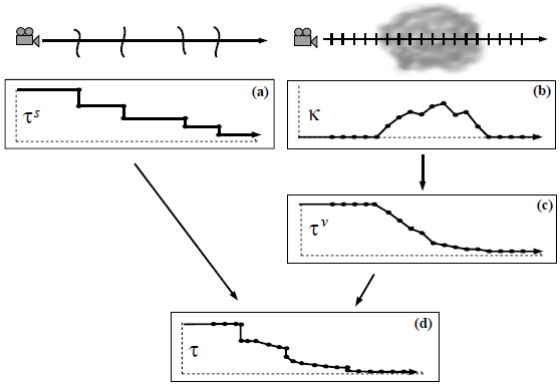
Pokud chceme získat deep shadow map, musíme zvolit sadu vzorků bodů, vybraných z celé roviny stínového obrazu. Pro každý vzorek se určí odpovídající funkce propustnosti, která popisuje úbytky světla zejména u primárního paprsku. Konečná funkce propustnosti je následně vypočtena, jako vážený průměr propustností funkcí v okolí vzorku bodů. Protože známe obrazový bod (x, y), mohou být vypočítány povrch a objemové prvky, které protíná primární paprsek. Povrchové křížení lze pak nalézt sledováním dráhy paprsku nebo obyčejným skenováním renderu za předpokladu, že všechny vlastnosti objektů mohou být vyhodnoceny kdykoliv. V bodě (x, y) je následně možné vyjádřit propustnost funkce, jako produkt funkce propustnosti povrchu τ^s a propustnosti objektu τ^v. Povrchová propustnost je odhadnuta pomocí všech křížení primárního paprsku s povrchem v bodě (x, y). Zasažená plocha zaznamenává hodnotu hloubky Z_i^s a neprůhlednost O_i. Následně jsou tyto plochy vynásobeny $1 - O_i$, čímž se po částech získá konstantní funkce. Pro odhad funkce propustnosti objemu musela být vzorkována atmosférická hustota podél paprsku v pravidelných intervalech. Každý vzorek získal díky úbytku světla podél paprsku hodnotu hloubky Z_i^v a *koeficient zániku* κ_i. Lineární interpolací těchto vzorků získáme funkci zániku κ (*extinction function*). Celkovou velikost světla pronikajícího danou hloubkou z reprezentuje vztah (2.9) [9].

$$\tau^v(z) = \exp(-\int_0^z \kappa(z')dz') \tag{2.9}$$

Jelikož funkce není částečně lineární, provádí se vyhodnocení propustnosti v každém vrcholu funkce zániku a lineární interpolace. Postupně se vypočítává propustnost jednotlivých lineárních segmentů s jejich následným složením. Sloučením a vynásobením povrchové a objemové složky získáme funkci $\tau(z)$ ilustrovanou obrázkem 2.14. Pro linearitu funkce je nutné vypočítat kombinace vrcholů τ^s a následně mezi nimi lineárně interpolujeme. Tím získáváme funkci viditelnosti (2.10) pro celý pixel [9].

$$V_i, j(z) = \sum_{k=1}^{n} w_k|\tau_k(z), \tag{2.10}$$

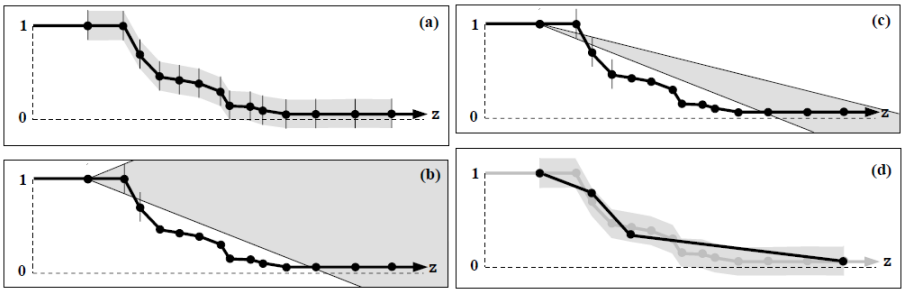
kde n je počet funkcí propustnosti na filtrovaném rozsahu, w_k je normalizovaný váhový filtr korespondující s bodem vzorku, τ_k je funkce propustnosti vrcholu.



Obrázek 2.14: Konstrukce funkce propustnosti. (a) Funkce povrchové propustnosti. (b) Funkce zániku. (c) Funkce objemové propustnosti. (d) Konečná funkce propustnosti [9].

Funkce viditelnosti může mít velký počet vrcholů (2.10) v závislosti na velikosti filtru a počtu vzorků na pixel. Z toho vyplývá, že je nutné u funkce provést kompresi. Stlačené funkce jsou uloženy jako pole dvojic s plovoucí desetinou čárkou. Obsahují informace o hodnotě a dílčí viditelnosti. Pro kompresi se obvykle používá kombinace chybové metriky (maximální chyba) a jednoduchého *greedy algoritmu* [9].

Greedy algoritmus (žravý algoritmus) je inkrementální. Postupně čte a zapisuje kontrolní body s rostoucí hloubkou z, přičemž vyžaduje pouze konstantní množství stavových informací. V každém kroku kreslí algoritmus pouze nejdelší úsečku, která se pohybuje v mezích chyby. Pro zjednodušení implementace se omezují výstupní hodnoty z podmnožinou vstupních hodnot z. Originální hodnoty aktuálního výstupního segmentu jsou (z_i', V_i'). V každém kroku je nutné zachovat povolený rozsah pro segment $[m_{lo}, m_{hi}]$. Pro další nový kontrolní bod (z_i', V_i') vstupní funkce zavádí omezení pro aktuální povolený rozsah tím, že nutí segment projít cílovým oknem definovaným z původních dvou bodů. Rozsah je inicializován pro všechna cílová okna v řadě, dokud není další krok prázdný (obrázek 2.15). Tento algoritmus je rychlý a jednoduchý na implementaci, ale vyžaduje neustálé ukládání hodnot segmentu [9].



Obrázek 2.15: Schéma algoritmu Greedy algoritmu. (a) Části lineární křivky a ilustrace vázané chyby. (b)Vrchol definující cílové okno. (c) Současný rozsah s protínáním cílového okna. (d) Výstupní segment rozšířený na aktuální hodnotu [9].

2.3 OpenGL

Informace pro tvorbu kapitoly byly čerpány z [12] a [1]. Podkapitoly se zaměřují na důležité prvky OpenGL, použité při tvorbě stínových map v této práci.

2.3.1 Framebuffer object (FBO)

Jako framebuffer objekty (FBO) jsou označovány OpenGL objekty umožňující vytvářet uživatelské framebuffery. Do těchto framebufferů lze provadét změny, které nenarušují obsah hlavního framebufferu. Funkce pro vytvoření, rušení nebo projení FBO jsou podobné, jako u klasických objektů OpenGL. Jako cílový parametr mohou být pouze hodnoty `GL_FRAMEBUFFER, GL_READ_FRAMEBUFFER` nebo `GL_DRAW_FRAMEBUFFER`.

FBO propojené s cílem dovolují používat určitou sadu bufferů. Proti výchozímu framebufferu, který má vyrovnávací paměti například jako `GL_FRONT` a `GL_BACK`, mají FBO vlastní sadu jmen obrazů. Každý obraz představuje bod propojení, umístěný ve framebuffer objektu, ke kterému je připojen. Existuje několik typů bodů připojení. Pro tuto práci je třeba především `GL_DEPTH_ATTACHMENTS` pro propojení obrazů ve formátu hloubky, kdy se z těchto obrazů stávají *Depth Buffery*.

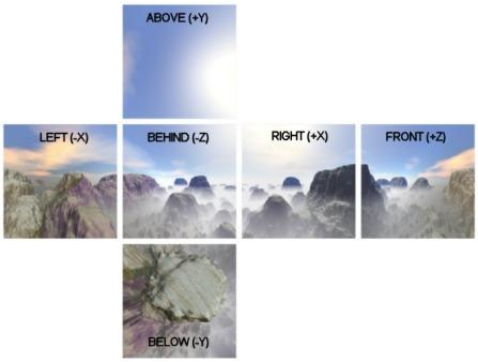
Pro připojení obrazu k FBO je nutné framebuffer object propojit s kontextem. Následně je možné připojit libovolný obraz z textury do framebuffer objektu. Textury mohou mít mipmapy. Každá mipmapa pak může obsahovat jeden nebo více jednoznačně identifikovaných obrazů. Například 1D textura obsahuje 2D obrazy s vertikální výškou 1. Pro modelování stínů v kubickém prostoru je potřeba minimálně šesti textur. Pro tento účel slouží tzv. *cube map*. Aby bylo možné s jednotlivým texturami cube mapy pracovat, je nutné použít *renderbuffer*.

Renderbuffer object obsahuje obraz a může být spojen pouze s konkrétním FBO. Tento buffer je především zaměřen a optimalizován na operace s pixely a může být u framebufferu využíván, jako cílové uložiště pixelů. Pro renderbuffer je nutné nejdříve vytvořit uložiště, kde je specifikován cílový renderbuffer a interní formát obrazů. Následně je možné renderbuffer spojit s vybraným FBO a nastavit požadovanou operaci nad pixely (draw, read, write). Uložení interního formátu pixelů poskytuje renderbufferu oproti klasické textuře výhodu v rychlosti. Hluboké stínové mapy vyžadují zápis informací o hloubce a barvě (i průhlednosti) stínu do dvou textur. Sekvenční zpracování textur není z hlediska doby zpracování příliš vhodné. Pro vyšší rychlost a efektivitu (zpracování informací o hloubce a barvě) se často využívá technika multiples rendering targets (MRT), dovolující zápis dat do několika textur současně.

2.3.2 Cube map

Pro vytvoření stínové mapy modelu bude použita cube map. Cube map je tvořena šesti vrstvami (2D obrazy), z nichž každá může být ještě tvořena několika úrovněmi minimap a využívá specifický typ `GL_TEXTURE_CUBE_MAP`. Jednotlivé stěny mají své označení 2.16 a lze k nim přistupovat pomocí `GL_TEXTURE_CUBE_MAP_(POSITIVE,NEGATIVE)_(X,Y,Z)`. Obrazy v cube mapě je možné jednoznačně identifikovat vrstvou a úrovní minimapy. Vrámci této práce bude použito několika cube map s rozdílným typem uložených informací. První typ map bude ukládat informace o hloubce (depth) objektů ve scéně. V aplikaci zastupují render buffer a k programu budou připojeny jako `DEPTH_COMPONENT`. Informace v těchto mapách budou rozhodovat o viditelnost bodů. Druhý typ map ukládá informace o barvě

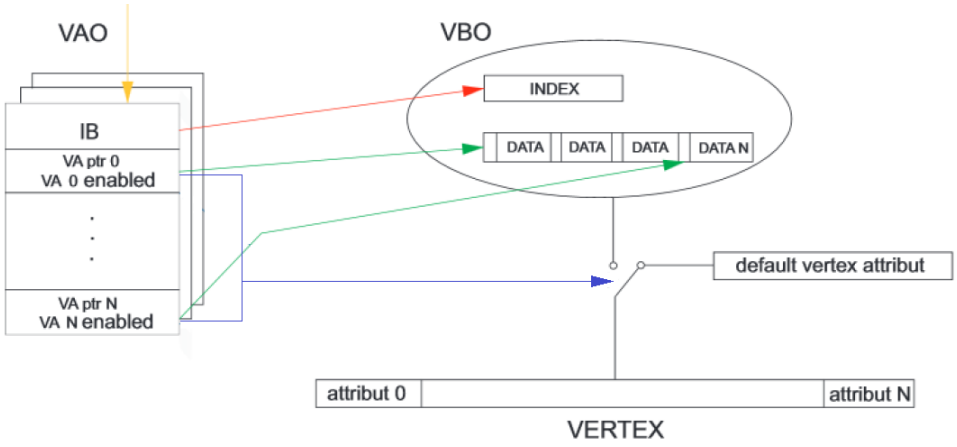
objektů viditelných z pozice světla. Všechny použité cube mapy budou podrobněji popsány v pozdějších kapitolách.



Obrázek 2.16: Cube map s označením stěn.
[10]

2.3.3 Vertex buffer object (VBO)

V OpenGL se pro uložení jednotlivých vrcholů používá tzv. vertex buffer. Jedná se o klasickou vyrovnávací pamět, do které je možné ukládat libovolné informace o vrcholu (např. pozici, barvu nebo normálový vektor). Aby vertex buffer správně pracoval je nutné ho spojit s *polem vertexů* (VA). Toto pole obsahuje všechny stavy potřebné pro vykreslení vertexových dat. Přístup k jednotlivým položkám (atributům) pole se nastavujeme pomocí *vertex attribut pointerů*. Současně VAP také popisují formát a interpretaci vertexových atributů, které jsou používány *vertex shaderem*.



Obrázek 2.17: Schéma funkce a propojení VBO, VAO a vector attribut pointer.

Na obrázku 2.17 je znázorněno zpracování vertexu a jeho atributů. V levé části obrázku se nachází vertex array object (VAO). Toto pole obsahuje odkazy na jednotlivé indexy vertexů a na jejich datové položky. Indexy vertexu bývají nejčastěji uloženy do struktury

[10] Převzato dne 13.1.2015 z: http://www.antongerdelan.net/teaching/3dprog1/multitex/multitextures.html

označované *index buffer* (IB). Tato vyrovnávací paměť umožnuje snadnou manipulaci s vrcholy. Použití IB má také zásadní vliv na množství dat ve VBO, každý vrchol je ve vertex bufferu zastoupen pouze jednou položkou. Pokud není index buffer využíván, mohou se některé vrcholy opakovat a tím zvyšovat paměťovou náročnost. V poli vrcholů (VAO) se kromě index buffer, ukazujícím na jednotlivé vrcholy ve vertex bufferu, obsahuje také reference na data (atributy) vertexu (VA ptr). Celkový počet počet odkazů je určen počtem povolených atributů vertexu (vertex attrib enabled). Výsledný vertex následně vstupuje do vertex shaderu se všemi svými povolenými atributy.

Kapitola 3

Návrh aplikace

Hlavním cílem této práce je vytvoření metody vykreslení stínů neprůhledných těles s využití hlubokých stínových map pro odstranění aliasu a následná implementace této metody do aplikace, zobrazující demonstrační scénu. Základní scéna obsahuje pouze informace o geometrii objektů a pozici jednoho zdroje světla (tzv. mesh). Tento světelný zdroj ale nevytváří žádné osvětlení scény. Nejdříve je nutné toto osvětlení vytvořit. Metod, řešících osvětlení, existuje velké množství. Jelikož tvorba osvětlení není hlavní náplní práce, bylo zvoleno osvětlení pomocí Lambertova modelu.

Aplikace bude realizovat stínování pomocí klasické stínové mapy 2.2.1 a pomocí hluboké stínové mapy 2.2.2. Hluboká stínová mapa (DSM) bude obsahovat celkem dvě vrstvy: první vrstva bude zaznamenávat informace o stínech neprůhledných těles a druhá ukládá data o hranách ošetřených proti aliasu. Stíny průhledných těles nebo barevné stíny nebudou touto DSM podporovány, ale bude je možné snadno doimplementovat v budoucím rozvoji práce. Každá vrstva bude realizována cube mapou 2.3.2 barvy a hloubky objektů z pozice světla. Vytváření a zpracování těchto map je realizováno mimo hlavní scénu. Pro tento účel bude použit framebuffer object 2.3.1. U stínových map bude nutné ošetřit nepříjemné jevy jako aliasing 2.2.1 nebo shadow acne 4.2, které mají velký vliv na kvalitu výsledných stínů. Obě metody stínování budou ošetřeny proti vzniku shadow acne, ale pouze DSM umožní odstranit i alias. Pro odstranění aliasu bude hluboká stínová mapa používat vrstvu, obsahující cube mapu s informacemi o hranách těles, které vrhají stíny tzv. okřídlené hrany obrázek 4.4), kde budou tyto hrany vykresleny s antialiasem a začlení se do výpočtu stínování scény. Typ stínování bude možné měnit za běhu programu. To zajistí snadné porovnání výsledků obou metod.

Kapitola 4

Implementace

Pro implementaci výsledné aplikace byl použit jazyk C++ spolu s nástrojem Microsoft Visual Studio 2008 (MVS). MVS zajišťuje snadnou manipulaci a vytvoření hlavního vykreslovacího okna, kompilaci aplikace a provázanost s OpenGL (s využitím knihovny **GLEW** [11]). Samotné vykreslení scény je následně plně v režii grafického adaptéru (GPU).

4.1 Struktura programu

Tato podkapitola popisuje celkovou strukturu programu. Úvod kapitoly se zaměří na hlavní funkce aplikace pro načtení, vytvoření a zpracovaní modelu. Dále bude provedeno detailní seznámení se strukturami, které používá program (především se bude jednat o struktury shaderu a FBO).

4.1.1 Hlavní funkce

Hlavní funkcí aplikace je funkce `main()`, nacházející se v souboru `Main.cpp`. V main funkci se nejdříve vytváří třída vykreslovacího okna. Tato třída je následně zaregistrována a jsou vypočítány rozměry okna spolu s rozměry klientské vykreslovací oblasti. Na základě těchto informací je pak vytvořeno a zobrazeno hlavní okno aplikace. Při změně velikosti okna je volána funkce `onResize()`, která přizpůsobuje velikost klientské oblasti novému rozměru okna. Jednotlivé verze OpenGL se liší v podporovaných primitivech a funkcích. Je proto nutné provádět test kompatability požadované verze OpenGL s podporovanou verzí OpenGL na grafickém adaptéru. Tím se zamezí především nežádoucím pádům aplikace a nekorektnímu vykreslení výsledné scény. Pokud je verze OpenGL podporována, začíná program načítat geometrii modelu funkcí `LoadWorld()`(tato funkce bude popsána později) a vytvoří se tzv.`mesh`. Z parametrů programu je zjištěno rozlišení stínových map a aplikace přechází do stavu inicializace objektů (`InitGLObject()`). Po úspěšné inicializaci všech potřebných objektů se spustí hlavní smyčka programu. Tato smyčka přijímá a reaguje na zasílané zprávy a v případě, že nedošlo k žádné události, volá funkci `OnIdle()`(zpracování a vykreslení scény). Při zavření okna nebo stisku klávesy ESC je hlavní smyčka ukončena a provede se úklid vytvořených objektů (`CleanupGLObjects()`).

[11] Dostupné z: http://glew.sourceforge.net/

4.1.2 Načítání geometrie modelu

O načtení geometrie z textového souboru se stará funkce `LoadWorld()`. Informace o názvu souboru jsou této funkci předány z příkazového řádku. Následně dochází k otevření souboru pro čtení a načítání jednotlivých dat. Činnost `LoadWorld()` lze popsat konečným automatem. První stav automatu zjišťuje převrácenost tělesa (`flip`). Pokud má těleso vlastnost flip, jedná se o duté těleso, které modeluje například krabici. Normály povrchu tělesa pak směřují dovnitř. Následně přechází automat do stavu načítání geometrie tělesa. Program podporuje pouze tři typy geometrie: kvádr (`box`), koule (`sphere`) a válec (`cylinder`). Každé těleso vyžaduje pro svou jednoznačnou specifikaci v prostoru scény zadání dalších povinných a nepovinných parametrů. Mezi povinné parametry patří zadání pozice tělesa ve scéně (`pos`), barva objektu (`col`) a informace pro převod tělesa na síť trojúhelníků (*teselace*). Do nepovinných parametrů zahrnuje aplikace transformace tvaru, velikosti a natočení tělesa. Podporovány jsou změna velikosti (scale - `scl`) a rotace `rot`. Některé parametry se mohou aplikovat pouze na určitou část tělesa. Příklad zápisu parametrů objektu je na obrázku 4.1.

```
box 1 1 1 {              // nastavení teselace objektu
  pos 0 7.85 0           // pozice tělesa ve scéně
  scl 0.1 0.15 0.1       // nastavení velikosti stran
  col 1 1 1 1            // barva kvádru (bílá)
  bottom-col 0 1 0 1     // barva spodní stěny bude zelená
  bottom-rad 0 0 1       // těleso bude zdroj modrého světla
}                        // zdroj se nachází na spodní straně
```

Obrázek 4.1: Příklad vytvoření tělesa typu kvádr jako zdroje modrého světla, nacházejícího se na spodní straně kvádru.

Ve scéně je potřeba vytvořit bodové osvětlení. Aplikace podporuje pouze jeden bodový zdroj, zastoupený libovolným podporovaným geometrickým tělesem. Ostatní světelné zdroje jsou vykresleny, ale nejsou zohledněny při výpočtu osvětlení scény. Pro určení směru a barvy osvětlení se využívá speciální parametr `bottom-rad` a `top-rad`.

Jednotlivá tělesa jsou pak na základě načtených dat převedena na reprezentaci pomocí trojúhelníků. Trojúhelník tvoří základní a nejjednodušší jednotku vykreslení objemových (3D) těles. Pro každý trojúhelník se vypočítávají pomocné informace o reflexi barvy, pozici středu a směru normály. Tyto informace se ukládají do struktury `TFaceProps` a lze je později využít například pro vytvoření *okřídlených hran*. Všechny trojúhelníky jsou tvořeny třemi vertexy. Vertexy obsahují nejen informace o pozici v prostoru scény, ale také například informace o normále vertexu a jeho barvě. Tato vedlejší data mohou být libovolná a závisí pouze na dalším použití vertexu. Pro základní vykreslení scény v této aplikaci je potřeba znát pozici, barvu a normálu vertexu. Celková velikost paměti pro uložení vertexu je pak rovna `10*float` (4 - barevné složky R,G,B,A , 3 - pozice x,y,z a 3 - normála x,y,z).

Funkce `LoadWorld()` následně vezme všechna načtená a vypočítaná data a vytvoří z nich strukturu typu mesh (4.1.3). Pokud nedojde při zpracování k chybovému hlášení, je z této meshe získána a samostatně uložena pozice bodu zdroje světla. Tento bod vyžaduje odlišný typ vykreslení, proto je nutné ho oddělit od hlavní geometrie.

4.1.3 Mesh

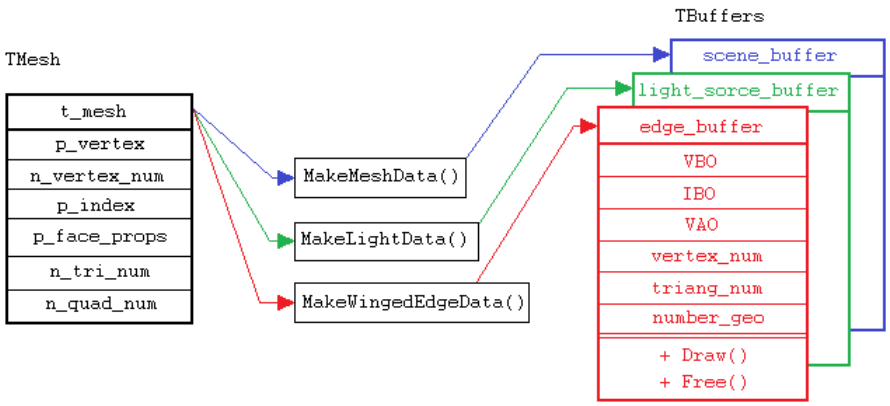
Hlavní strukturou pro uložení dat získaných při načítání modelu je struktura `Mesh`. Jedná se o centrální uložiště všech informací o modelu, ze kterého si následně funkce vybírají pouze potřebné informace. Například funkce `MakeWingedEdgeData()` si z meshe vybírá informace o pozici vertexů ale ignoruje informaci o jejich barvě. Mesh tedy přímo nenaplňuje vertex buffery daty, ale funguje jen jako prostřeník, poskytující data pro tyto buffery.

4.1.4 Struktura bufferů

Vykreslení scény se stínováním a osvětlením bývá výpočetně velmi náročné. Pokud bychom tyto výpočty scény nechali přímo provádět procesorem (CPU), docházelo by ke snížení výkonu procesoru a tím i celé aplikace. Procesor by musel neustále data načítat z paměti, zpracovávat je a následně výsledky odesílat pro zobrazení na grafický adaptér. To je ale velmi časově náročné a pro zpracování rozsáhlých scén zcela nevhodné řešení. V moderních počítačích je snaha využít vysokého výpočetního výkonu grafického adaptéru pro snížení celkové zátěže CPU. Oproti CPU je na grafickém adaptéru (GPU) implementována podpora OpenGL. Tato podpora umožňuje snadnější výpočet grafických informací a v kombinaci s rychlými vyrovnávacími paměťmi (buffery) a samotným výkonem GPU poskytuje velmi rychlé zpracování i složitějších scén bez snížení celkového výkonu počítače.

Aby mohla být data zpracována a vykonána grafickým adaptérem je potřeba uložit tato data do bufferů. Pro vytvoření a uchování bufferů se používá struktura `TBuffers`. Tato struktura obsahuje tři hlavní objekty: vertex buffer (VB) (2.3.3), index buffer (IB) a pole vertexů (VAO). Kromě těchto objektů se zde ukládájí i informace potřebné pro vykreslení objektu. S využitím těchto dat následně struktura `TBuffers` poskytuje funkci `Draw()` pro vykreslení do aktuálního framebufferu. Vykreslení může být dvojího typu. Pokud index buffer neobsahuje žadné indexy, vykresluje se pole vertexů pomocí `DrawArray()`. V opačném případě se kreslí jednotlivé elementy na základě indexů (`glDrawElement()`)

V rámci této práce se používá několik `TBuffers` struktur. První struktura s názvem `scene_buffer` je určena pro vykreslení celkové scény bez bodového zdroje světla. Bodový zdroj světla musí být uložen samostatně (struktura `light_source_buffer`), protože vyžaduje jiný typ vykreslení. Poslední používaná struktura ukládající hrany objektů je `edge_buffer`. Vytváření `TBuffers` je zobrazeno obrázkem 4.2.



Obrázek 4.2: Schéma tvorby TBuffers ze struktury TMesh.

4.1.5 Framebuffer object

Hluboká stínová mapa funguje na principu viditelnosti těles ve scéně. Stíny se pak následně vytváří na povrchu těles, která jsou vzhledem k pozici zdroje světla, v zákrytu jiných objektů scény. Než může deep shadow mapa vytvořit stínování scény, potřebuje znát hloubku (stejně jako klasická stínová mapa) a barvu těles ve scéně. Tyto informace se dají získat předvykreslením modelu. Aby výsledky vykreslení nezměnily obsah hlavního vykreslovacího okna (hlavního framebufferu), využívá se technika vedlejšího framebufferu (2.3.1). V tomto framebufferu se zpracovávají pouze pomocné výpočty. Při vykreslování výsledného stínování se potom vedlejší framebuffer odpojí a vykresluje se zpět do hlavního framebufferu. Pomocný framebufferu v aplikaci představuje struktura `FramebufferObject`. Tuto strukturu lze rozdělit na několik fází.

Nejdříve se provádí inicializace všech objektů, se kterými framebuffer pracuje nebo je dále poskytuje. Dochází k vytvoření nového framebuffer objektu a generaci cube map textur (2.3.2). Celkem se inicializují čtyři cube mapy odlišných typů a parametrů. První typ cube map, zastupující ve framebufferu render buffer, ukládá hodnoty hloubky (depth). Pro ukládání hloubky je nutné nastavit formát textury na `GL_DEPTH_COMPONENT`. Velikost uložených hodnot je dostačující v rozsahu `GL_FLOAT`. Další typ zaznamenává informace o barvě všech ploch objektů, viditelných přímo z pozice světla. Jedná se o klasický typ textury s kanály RGBA. Celková velikost kanálů je 24 bitů (8 bitů pro každý kanál). O nastavení jednolivých textur se stará funkce `DefineCubeTexture()`. Ta ze vstupních parametrů určí a vytvoří požadovaný typ cube map. OpenGL pracuje s cube mapou, jako s kolekcí šesti textur, proto je nutné nastavovat jednotlivé stěny cube mapy zvlášť.

Druhou fází framebufferu je připojení inicializovaných textur k programu. Cube mapy budou využívat oba framebuffery. Lišit se však bude uplatnění těchto map. Hlavní framebuffer potřebuje z cube map pouze číst data, zatímco vedlejší framebuffer vyžaduje zapisování výsledků pomocného vykreslení scény do cub mapy. Z tohoto důvodu se musí implementovat několik `Bind` funkcí. Struktura funkce Bind, pro zápis do textur, se skládá z několika kroků. Nejprve se připojí framebuffer, do kterého bude provedeno vykreslení (pro zápis vždy vedlejší framebuffer objekt). Nastavíme velikost oblasti uživatelského okna pro vykreslení na rozlišení stěny cube mapy funkcí `glViewport()`. Dalším krokem je připojení požadovaných textur. Připojení probíhá pomocí funkce `glFramebufferTexture2DEXT()` kdy nepřipojujeme celou cube map texturu, ale pouze její jednu stěnu, do které se má provést zápis. Současně se také stanoví i účel textury. Pro záznam informací o hloubce je typ připojení (attachment) nastaven na `GL_DEPTH_ATTACHMENT` a pro uložení barevných dat na `GL_COLOR_ATTACHMENT`. Framebuffer podporuje techniku MRT (multiple render targets). Tuto technika ušetří výpočetní čas aplikace, protože nebude nutné vykreslovat scénu ve dvou průchodech (jeden pro získání hloubky, druhý pro získání barvy) ale pouze v jednom průchodu. Na závěr nastavíme ještě vykreslovací buffer (`DrawBuffer`) a zkontroluje správnost připojení. U připojení textury cube mapy pro čtení stačí vytvořit a nastavit texturovací jednotku příkazem `glActiveTexture()` a požadovanou texturu připojit k této jednotce.

Posledním fází framebufferu je vymazání a uvolnění místa v paměti všech vytvořených objektů - funkce `Free()`. Tato funkce se zavolá při každém korektním i chybovém ukončení aplikace.

4.1.6 Shader

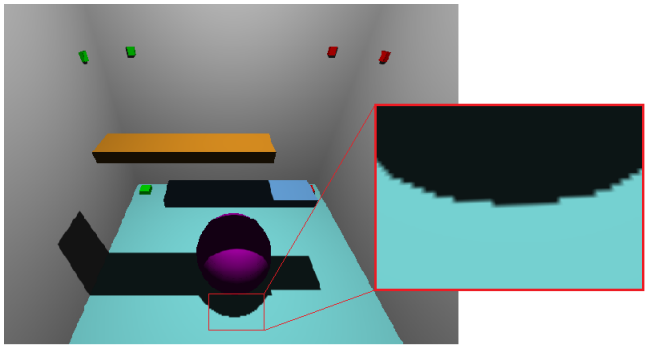
Jako *shader* [1] bývá v počítačové grafice označován uživatelem definovaný program, který je určen pro běh ve specifické fázi grafického procesoru. Obecně existuje několik druhů shaderů.

Mezi hlavní shadery patří vertex a fragment shader. Vertex shader (VS) zpracovává každý vertex vstupní geometrie zvlášť. Tento vertex pak libovolně transformuje (v této aplikaci bude VS především vrcholy násobit maticí pohledu), ale nikdy nemůže vertex odstranit. Fragment (pixel) shader provádí modifikace barvy pixelů 2D obrazu scény nebo aplikaci textur na tyto pixely.

Struktura `Shader` pracuje pouze s dvěmi shadery - pixel a vertex shader. Samotná generace těchto shaderů probíhá ve funkci `Compile()`. Nejprve dochází k vytvoření shaderů (`glCreateShader()`). Dále jsou pixel a vertex shader zkompilovány. Pokud nedojde při kompilaci k chybě, vytváří se hlavní shader (`program_object`). Následně se k vytvořenému `program_object` připojí vertex shader i pixel shader spolu s jejich atributy. Na závěr se program funkcí `glLinkProgram()` spojí se svými komponentami a je připraven k použití. Aplikace pracuje hned s několika shadery, kdy každý z těchto shaderů plní specifický úkol. Pomocí shaderu se v aplikaci například vytváří osvětlení nebo stínování.

Shader stínové mapy

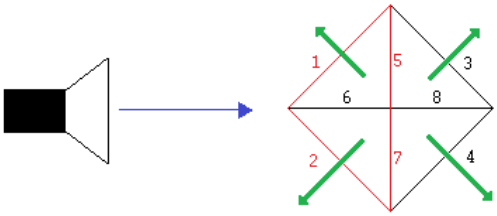
`ShadowMapShader` reprezentuje stínování pomocí klasické stínové mapy. Shader potřebuje pro svou funkci stínovou mapu. Stínová mapa se získá vykreslením scény s využitím shaderu `ShadowShader`. Samotné stíny pak vznikají pouze ve fragment shaderu. V hlavní funkci se nejdříve zavolá funkce `DepthCompare` pro testování viditelnosti pixelu. Funkce `DepthCompare` vypočítá vzdálenost pixelu od zdroje světla a tuto vzdálenost následně porovná s hodnotou ve stínové mapě. Výsledek porovnání nabývá hodnot 0.0 (pixel je ve stínu) nebo 1.0 (pixel osvětlen). Tato hodnota je pak vrácena hlavní funkci, kde se začlení do výpočtu barvy pixelu. Vykreslení stínování scény pomocí `ShadowMapShader` je na obrázku 4.3.



Obrázek 4.3: Stínování scény pomocí klasické stínové mapy s ukázkou vzniklého aliasu.

Shader okřídlených hran

Na obrázku 4.3 je patrný vznik aliasu při používání stínové mapy. Kvalita stínů není proto příliš velká. U hluboké stínové mapy bude potřeba artefakty aliasu eliminovat a vytvořit kvalitní stínování. Hrany objektů, které vrhají stíny, bude nutné vykreslit s antialiasem (v OpenGL technika GL_LINE_SMOTH), což povede k zlepšení výsledné kvality stínů. Pro zjištění těchto hran objektů se v aplikaci používá metoda *okřídlených hran* (Winged edge) [4]. Princip metody je zobrazen na obrázku 4.4.



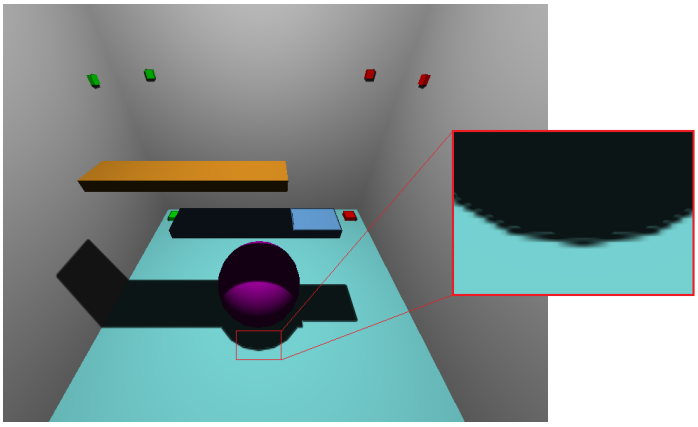
Obrázek 4.4: Princip okřídlených hran.

Na obrázku 4.4 je zobrazen čtyřboký jehlan. Světle modrá barva reprezentuje směr pohledu kamery. Zelenou barvou jsou znázorněny normály stěn jehlanu. Obecně lze hranu označit jako okřídlenou, pokud pro stěny jehlanu svírajících tuto hranu platí, že jedna stěna je viditelná a druhá neviditelná z pohledu kamery. Matematicky můžeme tuto vlastnost vyjádřit následovně: Pohled kamery zastupuje vektor P a normály stěn jsou označeny jako N_1 a N_2. Potom jako okřídlené hrany označuje pouze ty hrany, pro které je výsledek rovnice $(P \cdot N_1) * (P \cdot N_2)$ záporný.

Stejně zjisťuje okřídlené hrany i shader `WingedEdgeShader`, kde je jediným rozdílem nahrazení normálových vektorů stěn rovinami, ve kterých tyto stěny leží a umístění kamery na pozici zdroje světla. Výpočet se provádí ve vertex shaderu. Pokud vertex splňuje podmínku (tzn. hrana, které vertex náleží, je okřídlená), vypočítá se pozice vertexu ve scéně. Výsledky `WingedEdgeShader` shaderu jsou zapsány do dvou cube map. První mapa ukládá depth hodnoty hran a druhá mapa zaznamenává vykreslení hran bílou barvou.

Shader hluboké stínové mapy

Pro vytvoření stínů bez aliasu se používá shader `DeepShadowMapShader`. Shaderu potřebuje celkem tři cube mapy. Kromě stínové mapy scény se k shaderu připojují ještě cube mapy hloubky a barvy okřídlených hran získáných při vykreslení scény pomocí `WingedEdgeShader` shaderu. Podobně, jako u stínování s klasickou stínovou mapou, se ve fragment shaderu volá funkce `DepthCompare()`. Zde se provede výpočet vzdálenosti pixelu od zdroje světla a porovnání této vzdálenosti s hodnotou ve stínové mapě. Pokud je pixel ve stínu neprůhledných těles je vrácena hodnota 0.0. V opačném případě se provádí test okřídlených hran. Z cube mapy s hloubkou okřídlených hran je zjištěna hodnota hloubky a následně je tato hodnota porovnána s hodnotou vzdálenosti pixelu od zdroje světla. Pokud se pixel nachází v zákrytu okřídlené hrany, vrací se hodnota aplha kanálu cube mapy s barvou okřídlených hran. Konečný výsledek vykreslení scény je na obrázku 4.5.

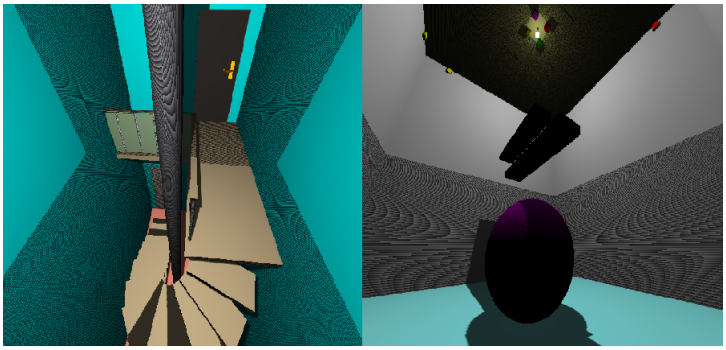


Obrázek 4.5: Stíny vytvořené hlubokou stínovou mapou.

4.2 Řešení problémů při stínování

Při vytváření stínování vzniklo několik problémů, které bylo nutné řešit. Tyto problémy byly většinou způsobeny nedokonalostmi metod použitých v rámci práce.

Hlavním problémem, který se velmi často projevuje při stínování, je tzv. *shadow acne* [13]. Shadow acne se objeví, pokud porovnáváme přímo hodnotu hloubky uloženou ve stínové mapě s hloubkou pixelu, kdy dochází k self-shadowing (objekt vrhá stín sám na sebe). Tento problém demonstruje obrázek 4.6.



Obrázek 4.6: Scéna neošetřená proti shadow acné.

Tento problém je v práci odstraněn posunutím offsetu objektů (`PolygonOffsetFill`). Posunutím offsetů polynomů se docílí odstranění shadow acne, nicméně se současně posunují i stíny těles, což není žádoucí.

4.3 Model scény

Při vytváření scény je nutné dodržet požadavek na uzavřenost scény, jinak by se některé stíny těles promítaly do nekonečné hloubky scény. Současně i některé metody osvětlení

[13] Dostupné z: http://www.digitalrune.com/Support/Blog/tabid/719/EntryId/218/Shadow-Acne.aspx

vycházejí z předpokladu, že světlo scénu neopouští (například radiosita). Z těchto důvodů byl, jako výchozí model scény, zvolen cornell box. Cornel box [6] je zkušební 3D model pro testování správnosti zobrazení. Samotný cornel box zastupuje jedinné těleso typu `flip box`. Další objekty se většinou umisťují dovnitř tohoto boxu.

Pro důkladnější testování stínů byl vytvořen model schodiště. Tento model zobrazuje reálnou místnost domu, kde se nachází přízemí a první patro. Dominantou scény je schodiště otáčející se kolem sloupu. Zdroj světla leží přímo nad schodištěm. Díky velkému množství objektů ve scéně dochází k pomalejšímu vykreslení scény.

4.4 Ovládání aplikace

Pro spuštění vyžaduje aplikace dva parametry. První parametr udává zvolenou geometrii scény. Na výběr máme `cornell_box.txt` a `stairs.txt`. Druhý parametr stanový rozlišení stínové mapy. Doporučené a otestované rozlišení je 512x512. Zadání parametrů pak může vypadat následovně: `cornell_box.txt 512`. Po zadání parametrů lze aplikaci spustit. Dojde k vytvoření nového okna s vykreslenou scénou. Ve scéně je možné se libovolně pohybovat. Směr pohybu je vztažen k aktuálnímu pohledu kamery. Pohyb lze uskutečnit pomocí kláves:

- **Nahoru** - klávesy W a UPARROW
- **Dolů** - klávesy S a DOWNARROW
- **Vlevo** - klávesy A a LEFTARROW
- **Vpravo** - klávesy S a RIGHTARROW

Klávesa SPACE přepíná mezi stínováním pomocí klasické a hluboké stínové mapy. Při pohybu myši po okně aplikace dochází ke změně pohledu kamery. Aplikaci lze ukončit stisknutím klávesy ECS nebo zavření vykreslovacího okna.

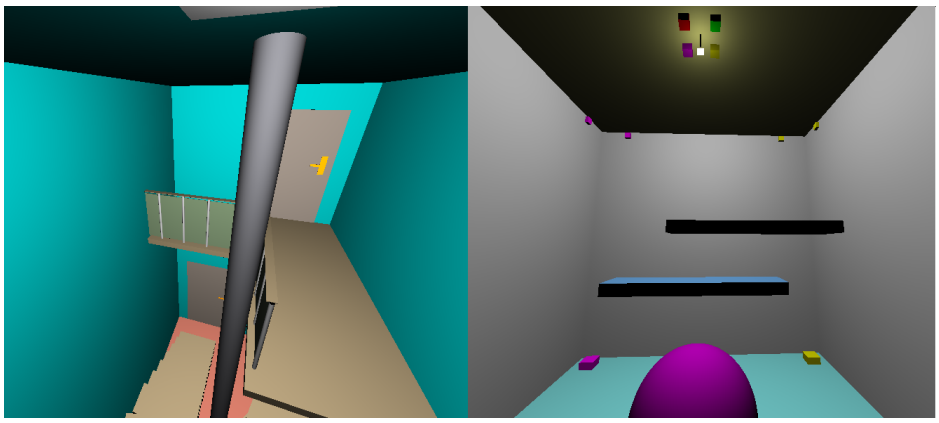
Kapitola 5

Dosažené výsledky

V této kapitole se nachází všechny dosažené výsledky. Úvod kapitoly se zaměří na použité osvětlení scény. Následně se vyhodnotí a porovnají výsledky metod stínování Shadow Map a Deep Shadow Map.

5.1 Osvětlení

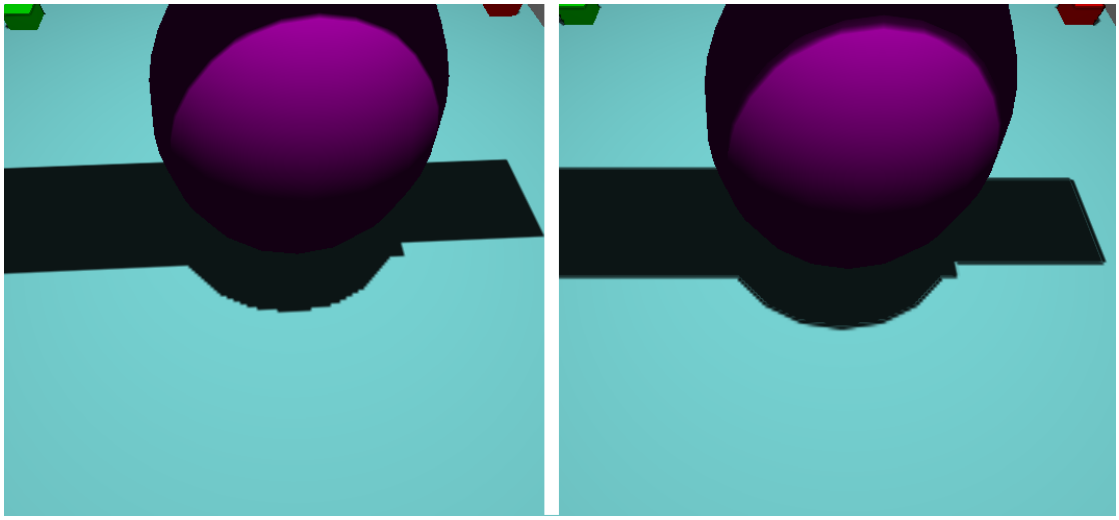
Pro osvětlení scény byl použit Lambertův model 2.1.2. Tento model patří k lokálním modelům a je velmi jednoduchý pro implementaci. Výsledné osvětlení nedosahuje vysoké kvality, ale pro aplikaci je dostačující. V dalším rozvoji práce bude tento jednoduchý lokální model nahrazen globalním osvětlením pomocí metody Light Propagation Volumes [8].



Obrázek 5.1: Osvětlení scény pomocí Lambertova modelu.

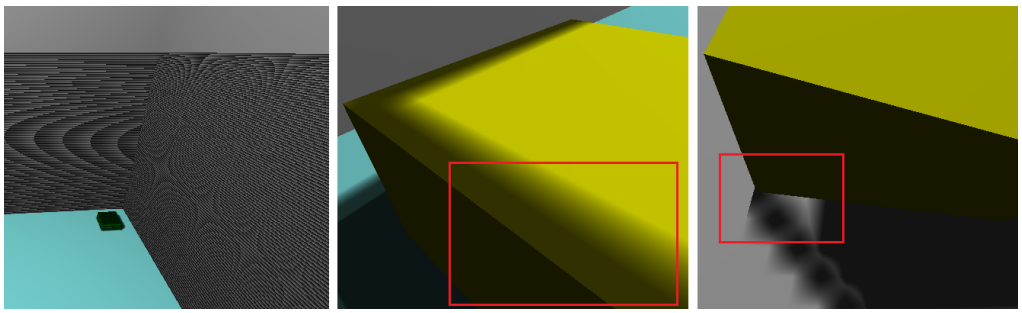
5.2 Stínování

Výstupy metod stínování jsou zobrazeny na obrázku 5.2. Z obrázku je patrné, že hluboká stínová mapa dosahuje lepších výsledků, než klasická stínová mapa. U hluboké stínové mapy je problém aliasu odstraněn, zatímco u klasické stínové mapy tento problem zůstává a je patrný například u stínu koule.



Obrázek 5.2: Výsledky stínování pomocí klasické stínové mapy (vlevo) a hluboké stínové mapy (vpravo).

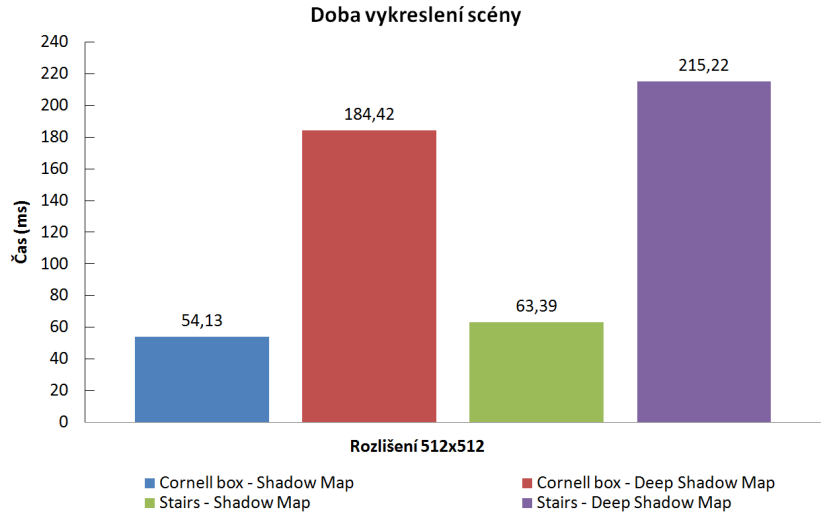
Při přiblížení pohledu jsou však na tělesech a stínech patrné nepříjemné artefakty. Většina těchto artefaktů vznikla při odstraňování základních nedostatků stínových map (obrázek 5.3). První nedokonalostí, která se velmi často vyskytuje u stínových map, je sebezastínění (označované též jako shadow acne nebo self-shadowing 4.2). Sebezastínění je způsobeno nepřesností při porovnání hloubky fragmentu s hloubkou ve stínové mapě, které následně vede k špatnému výsledku tohoto porovnání. Pro odstranění shadow acne se používá technika posunutí těles o malou hodnotu dozadu (o tzv. BIAS). Sebezastínění je sice zcela odstraněno, ale současně s posuvem těles se posouvají i jejich stíny (tzv. Peter Pan effect - obrázek (uprostřed)). Posuv se s rostoucí vzdáleností pixelu od zdroje světla zvětšuje. Řešení bývá optimalizace výpočtu hodnoty BIAS, kdy se tato hodnota mění na základě vzdálenosti pixelu od zdroje světla. Poslední negativní jev se projevuje při použití hran s odstraněným aliasem. Hrany, ošetřené proti aliasu, nemají posunutý offset. Při vykreslování stínů pak dochází v místě hrany k drobnému zastínění povrchu tělesa.



Obrázek 5.3: Nepříjemné jevy vzniklé při práci se stínovou mapou : (vlevo) sebezastínění pixelů, (uprostřed) drobné zastínění těles vlivem antialiasových hran a (vpravo) nedoléhající stíny vlivem posuvu stínů těles.

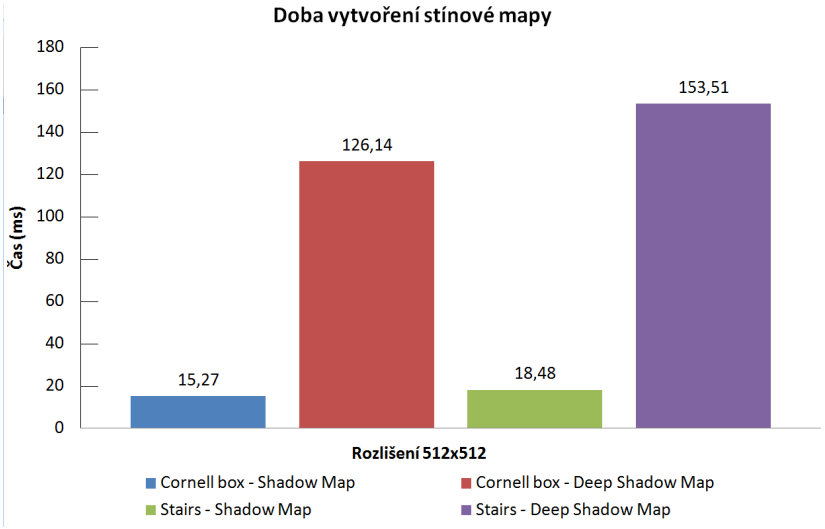
5.3 Rychlost aplikace

Na obrázku 5.4 je znázorněn graf celkové rychlosti vykreslení scény. Při použití klasické stínové mapy se scéna vykreslí mnohem rychleji, než scéna s hlubokou stínovou mapou.



Obrázek 5.4: Porovnání rychlosti vykreslení scény obou modelů.

Z porovnání rychlosti vykreslení DSM a SM (obrázek 5.5) je patrné, že doba zpracování je u hluboké stínové mapy vyšší. Důvodem je především nutnost výpočtu a vykreslení okřídlených hran. Je zde patrná i závislost rychlosti na složitosti scény, kdy vykreslení modelu cornell box trvá kratší dobu (model stairs je složen z více trojúhelníků).



Obrázek 5.5: Porovnání rychlosti výpočtu stínové a hluboké stínové mapy. Rychlost zpracování mapy je ovlivněna složitostí scény.

Kapitola 6

Závěr

Cílem práce bylo odstranění aliasu stínů u stínových map pomocí antialiasu hran. Pro účely vytvoření aplikace bylo nutné nastudovat základní principy tvorby stínových map spolu s řešením aliasu stínů v OpenGL a seznámit se s modely osvětlení scény. Na základě získaných informací byla implementována výsledná aplikace. Osvětlení je realizováno bodovým zdrojem světla, který reprezentuje Lambertův model osvětlení. Aplikace podporuje stínování pomocí klasické stínové mapy (Shadow Map) a hluboké stínové mapy (Deep Shadow Map (DSM)). Deep Shadow Map nezobrazuje stíny průhledných objektů ani nevytváří barevné stíny. Pro odstranění aliasu stínů hluboká stínová mapa vykresluje hrany těles s antialiasem. K zjištění hran, které mají být ošetřeny proti aliasu, DSM používá techniku okřídlených hran (Winged Edge).

Při vytváření stínování scény se narazilo na několik problémů. Prvním problémem bylo zobrazení self-shadowing. Problém byl odstraněn pomocí posuvu těles směrem dozadu. Tím se však projevil další nepříjemný jev - posunutí stínů. Ani vykreslování hran s antialiasem neproběhlo bez problémů. Jelikož jsou tyto hrany vykresleny bez posunutí offsetu, dochází k drobnému zastínění těles. Všechny problémy bude nutné v dalším pokračování práce odstranit.

Hluboká stínová mapa v této práci pracuje pouze s dvěmi vrstvami stínů (stíny neprůhledných těles a stíny okřídlených hran). Proto se další rozvoj práce zaměří na zvýšení počtu vrstev DSM, které umožní zobrazit barevné stíny průhledných těles. Vzhledem k faktu, že stíny stále obsahují nežádoucí artefakty, bude nutné optimalizovat vypočet stínování. Ani osvětlení scény není příliš kvalitní. Jedním z možných řešení by mohlo být použití globálního modelu osvětlení (např. Light Propagation Volumes).

Literatura

[1] OpenGL Wikipedia [online]. `https://www.opengl.org/wiki/`, 2015 [cit. 2015-01-24].

[2] Lambert's cosine law [online]. `https://www.princeton.edu/ achaney/tmve/wiki100k/docs/`, [cit. 2015-01-24].

[3] Ahokas, T.: *Shadow Maps*. Helsinky Univerzity of Technology, Telecommunications Software and Multimedia Laboratory, Tik-110.500 Seminar on Computer Graphics, 2002.

[4] Baumgart, B. G.; Latecki, L. J.; Petitjean, S.: *A polyhedron representation for computer vision*. Proceedings of the May 19-22, 1975, national computer conference and exposition on - AFIPS '75 [online]., 1975, 10.1007/3-540-60477-4_11.

[5] Buss, S.: *3-D computer graphics a mathematical*. Cambridge: Univerzity Press, první vydání, 2003, 371 s., iSBN 80-85615-77-0.

[6] Coral, C. M.; Torrance, K. E.; Greenberg, D. P.; aj.: *Modeling the interaction of light between diffuse surfaces: a monthly publication of the ACM Publications Office*, ročník 18 issue 3. New York: Association for Computing Machinery, 1984, iSBN 10.1145/964965.808601.

[7] Eisemann, E.; Assarsson, U.; Schwarz, M.; aj.: *Casting Shadows in Real Time. ACM SIGGRAPH ASIA 2009 Courses on - SIGGRAPH ASIA '09*. New York: Association for Computing Machinery, 2009, iSBN 10.1145/965139.807402.

[8] Kaplanyan, A.; Dachsbacher, C.: *Cascaded Light Propagation Volumes for Real-time Indirect Illumination. In Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, I3D '10*. New York: Association for Computing Machinery, 2010, 99-107 s., iSBN 978-1-60558-939-8.

[9] Lokovic, T.; Veach, E.: *Deep shadow maps. Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*. New York: Association for Computing Machinery, 2000, iSBN 10.1145/344779.344958.

[10] Phong, B. T.: *Illumination for computer generated pictures: a monthly publication of the ACM Publications Office*. New York: Association for Computing Machinery, 1975, iSBN 10.1145/360825.360839.

[11] Schwenk, K.: *A Survey of Shading Models for Real-time Rendering: How to steal components from various BRDFs and combine them into your own [online]*. 2011 [cit. 2015-01-24], 43 s.

[12] Segal, M.; Akeley, K.: OpenGL 4.5 Core Profile [online]. `https://www.opengl.org/registry/doc/glspec45.core.pdf`, 2014 [cit. 2015-01-24].

[13] Williams, L.: *Casting curved shadows on curved surfaces. SIGGRAPH '78 Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, ročník 12 issue 3. New York: Association for Computing Machinery, 1978, 270-274 s., dOI 10.1145/965139.807402.

[14] Žára, J.: *Moderní počítačová grafika*. Brno: Computer Press, první vydání, 2004, 609 s., iSBN 05-218-2103-7.

Seznam příloh

- Příloha 1: DeepShadowShader (pouze fragment shader)
- Příloha 2: WingedEdgeShader
- Příloha 3: Manuál aplikace
- Příloha 4: Plakátek

Příloha 1: DeepShadowShader

```
version 330
in vec4 ex_Color; // barva vertexu
in vec3 out_Normala; // normála vertexu
in vec3 toLight; // vzdálenost od zdroje světla
out vec4 out_Color; // výsledná barva pixelu
uniform samplerCubeShadow depth_map; // sampler stínové mapy
uniform samplerCubeShadow depth_edge_map; // sampler hloubky okřídlených hran
uniform samplerCube alpha_edge_map; // sampler barvy okřídlených hran
uniform vec3 light_col; // barva světelného zdroje
const float f = 100.0; // nastavení hodnoty far
const float n = 0.01; // nastavení hodnoty near

/* FUNKCE PRO VÝPOČET VZDÁLENOSTI BODU OD ZDROJE SVĚTLA */

float VectorToDepthValue(float f_bias) {
    float z = max(max(abs(toLight.x), abs(toLight.y)), abs(toLight.z)) - f_bias;
    float ndc_z = (f+n) / (f-n) - 2.0 * f * n / (z * (f-n));
    return ndc_z * 0.5 + 0.5;
}

/* TEST PIXELU - BUĎ JE PIXEL NA SVĚTLE NEBO JE VE STÍNU */

float DepthCompare (float f_bias) {
    float shadow_opacity = 0.0;
    float distance = VectorToDepthValue(f_bias * 0.0);
    //získaní vzdálenosti pixelu od světla
    float shadow = texture(depth_map, vec4(toLight, distance));
    // porovnání vzdálenosti pixelu od svět. zdroje a hodnoty ve stínové mapě
    if(shadow == 0.0)
        shadow_opacity = 0.0;
     else {
       float shadow_edge = texture(depth_edge_map, vec4(toLight, distance));
       // porovnání vzdáleností pixelu a hodnoty v cube mapě okřídlených hran
        if(shadow_edge < 1.0)
            shadow_opacity = 1.0 - texture(alpha_edge_map, toLight).w;
        else {
            shadow_opacity = 1.0;
        }
        shadow_opacity = min(shadow_opacity * shadow_opacity, shadow);
    }
    return shadow_opacity; // vrací se hodnota zastínění
}
```

```
/* HLAVNÍ FUNKCE - VYHODNOCENÍ PIXELU A NASTAVÍ BARVY PIXELU */

void main(void) {
    vec3 Normalized = out_Normala;
    vec3 toLightn = normalize(toLight);
    Normalized = normalize(Normalized);
    float shadow = DepthCompare(clamp(0.2*tan(acos(clamp(dot(toLightn, Normalized),
                                        0.0, 1.0))), 0.0, 1.0));
    // výpočet hodnoty zastínění pixelu
     out_Color = max(0.1, dot(toLightn, Normalized) * shadow) * ex_Color *
                            vec4(light_col, 1.0);
    // nastavení barvy pixelu
}
```

Příloha 2: WingedEdgeShader

VERTEX SHADER

```
version 330
in vec3 in_Position; // pozice vertexu
in vec4 in_Plane0, in_Plane1; // roviny hrany
uniform vec3 camera_pos; // pozice kamery
uniform mat4 t_modelview_projection_matrix; // projekční matice

/* HLAVNÍ FUNKCE - TEST OKŘÍDLENÝCH HRAN*/
void main(void) {
    float visibility0 = dot(in_Plane0, vec4(camera_pos, 1.0));
// test pozice kamery a první roviny vertexu
    float visibility1 = dot(in_Plane1, vec4(camera_pos, 1.0));
// test pozice kamery a druhé roviny vertexu
    if(visibility0 * visibility1 > 0.0)
// pokud není hrana okřídlená
        gl_Position = vec4(1.0, 0.0, 0.0, 0.0);
// odeslání vertexu do nekonečna
    else
        gl_Position = t_modelview_projection_matrix * vec4(in_Position, 1.0);
    // výpočet pozice vertexu ve scéně
}
```

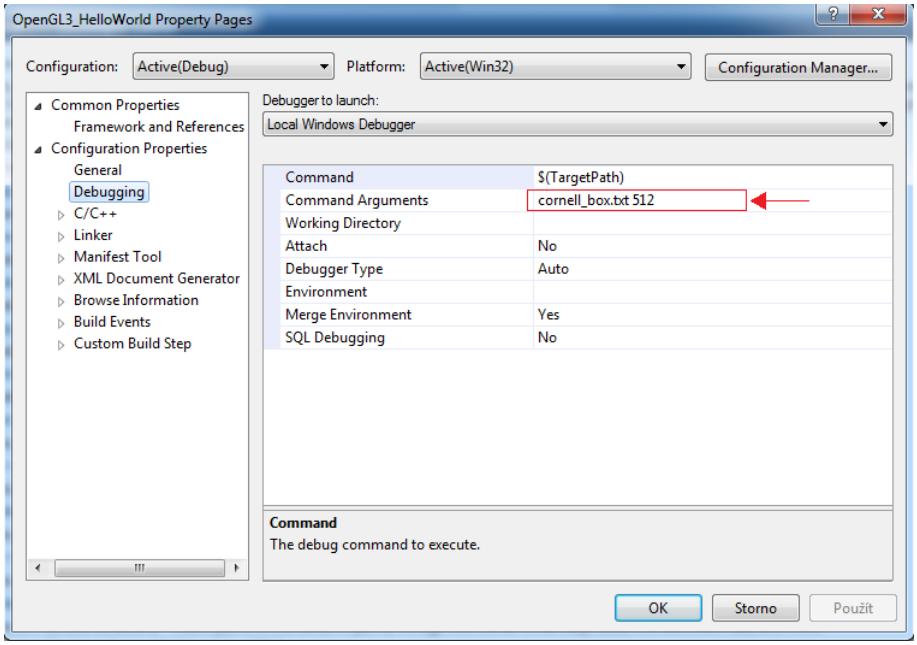
FRAGMENT SHADER

```
version 330
in vec3 out_Color;

/* HLAVNÍ FUNKCE - NASTAVÍ BARVY PIXELU */
void main(void) {
    out_Color = vec4(1.0, 1.0, 1.0, 1.0); //nastavení barvy pixelu na bílou
}
```

Příloha 3: Manuál aplikace

1. Zabalený archiv s aplikací rozbalíme do libovolné složky.
2. Spustíme program Microsoft Visual Studio 2008.
3. V hlavní nabídce programu zvolíme $File -> Open -> Project/Solution$.
4. Vyhledáme složku, kde je rozbalený archiv aplikace, a zvolíme soubor `DeepShadowMaps.sln`.
5. Pokud nedojde k chybě, zobrazí se v hlavní liště programu záložka *Project*.
6. Otevřeme záložku $Project -> Properties$
7. Zobrazí se nové okno. V levé části okna postupně rozklikneme $Configuration\ Properties -> Debugging$.
8. Následně na pravé straně okna zapíšeme do *Commands Arguments* název souboru modelu (cornell_box.txt nebo stairs.txt) a požadované rozlišení stínové mapy (testováno a doporučeno je rozlišení 512x512) a potvrdíme zvolené parametry kliknutím na OK. Příklad vyplnění řádku s argumenty je na obrázku níže.



9. Aplikaci nyní můžeme spustit pomocí $F5$
10. V nově vytvořeném okně je vykreslena zvolená scéna. Pohybování ve scéně lze provádět v závislosti na pohledu kamery klávesami (W, UPARROW) pro pohyb směrem vpřed, (S,DOWNARROW) směrem dozadu, (A, LEFTARROW) vlevo a (D, RIGHTARROW) vpravo. Přepínání mezi typem stínování je realizovánou klávesou SPACE.
11. Ukončení aplikace provádí klávesa ESC.

Příloha 4: Plakátek